RasPi

DESIGN
BUILD
CODE

24

Get hands-on with your Raspberry Pi

+

CONTROL FIREWORKS WITH A PI

BUILD A PI GLOVE

Plus Learn to code in FUZE BASIC

Welcome

From smartwatches to VR headsets, wearables are a huge story in tech right now, whether you're using them to track your activity or to control programs and games. In this issue, our expert guide explains how you can make your very own wearable in the shape of a Raspberry Pi glove. You'll learn how to build and program it to play music and use social media, so you'll be navigating with Minority Report-style gestures in no time!

Elsewhere in the issue, Chris Osborn explains how he's made the Raspberry Pi pop by using it to control a fireworks display. Plus we'll introduce you to FUZE BASIC and how to code a simple game, and take a look at how the Pi can serve up RSS feeds. Enjoy the issue!

April

Editor

Get inspired

Discover the RasPi community's best projects

Expert advice

Got a question? Get in touch and we'll give you a hand

Easy-to-follow guides

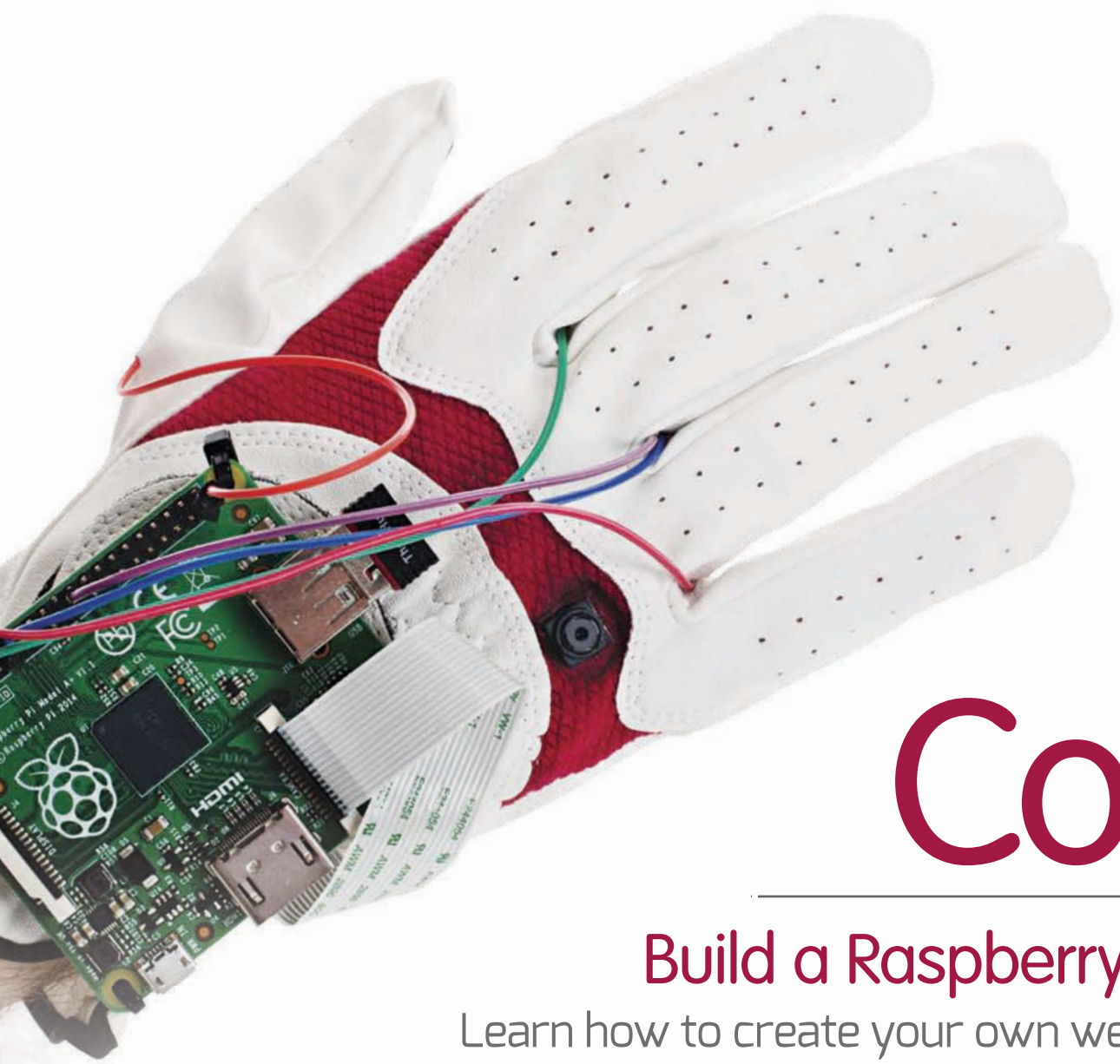
Learn to make and code gadgets with Raspberry Pi

From the makers of

LinuxUser & Developer

Join the conversation at...

@linuxusermag | Linux User & Developer | RasPi@imagine-publishing.co.uk



Contents

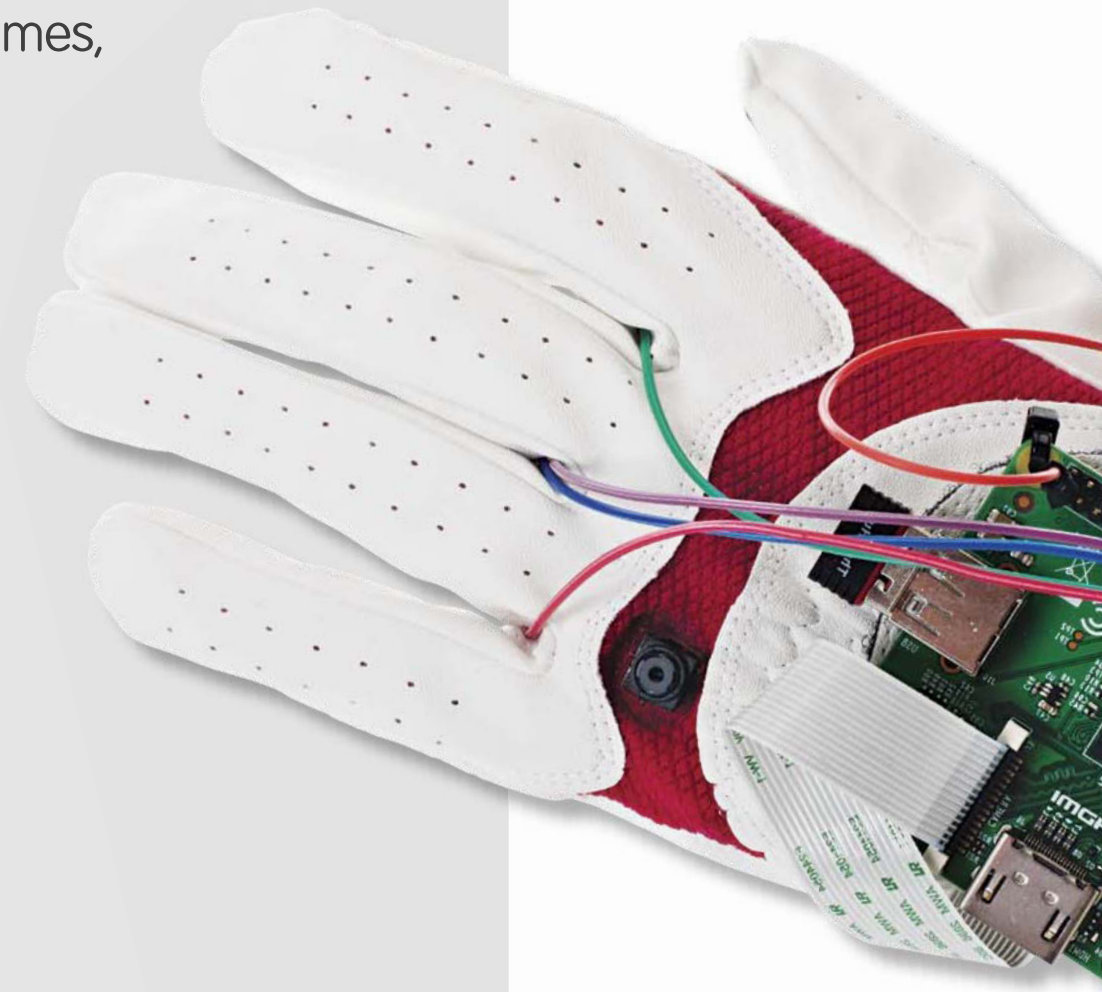
Build a Pi Glove

Create your own Wearable Tech glove that places control at your fingertips

The Pi Glove is our project name for a wearable, social media-controlling glove – we're going to show you how to build it and how to program it. With the advent of Google Glass, Android and Apple smartwatches and various other disruptive technologies, the devices we're beginning to use today point towards a future where we may well one day wear all of the gadgets that enable us to communicate with each other (before they are simply embedded into our bodies…). The more natural or ergonomic this medium becomes, the more likely we are to use it, so we're going to use a Raspberry Pi to create a powerful yet comfortable glove. In the first section of the tutorial we'll cover the hardware setup so that you can make your own version of the Pi Glove. In the second, we will cover the software to control music, a camera and social media using your glove's buttons.

THE PROJECT ESSENTIALS

Raspberry Pi A+
Clothes poppers
USB Wi-Fi dongle
Small cable ties
Glove
USB battery charger
Pi camera module



01 Strip and prepare the wires

Take the female-to-female jerky wires, select one end and remove the plastic coating; this can be done by applying a little pressure on the plastic cover – you can use your teeth but it's better to use a set of pliers! The end will now consist of a small metal spike. Prepare the other wires using the same method.

02 Attach the wires to the poppers

The next part is a little fiddly, so take your time as you work through each of the wires. First, you need to connect the wire to the popper whilst pinning the other end through the finger of the glove. Once through, close the two sides of the popper together, ensuring that the wire is secured in-between. This is easier if you turn each glove finger inside-out and then push the popper through. The key part of this step is to ensure that the wire stays connected.

03 Check the wires are in place

One wire/popper combo is attached to each of the four fingers and the thumb of the glove. The thumb forms the earth, or ground, of the connection when it comes into contact with one of the other metal finger poppers.

04 Add the Pi camera

The Pi camera is mounted into the glove, which enables you to angle your wrist and take a picture in the direction that the hand is facing. Make a small hole in the top of the glove, big enough for the camera lens to push through. On the Pi camera board you will find four small holes, so use these and the tiny cable ties to secure the camera onto the fabric of the glove.

Wpa_supplicant

Wpa_supplicant is a background program that runs and acts as the component to control wireless connection. It supports both text and GUI interfaces.

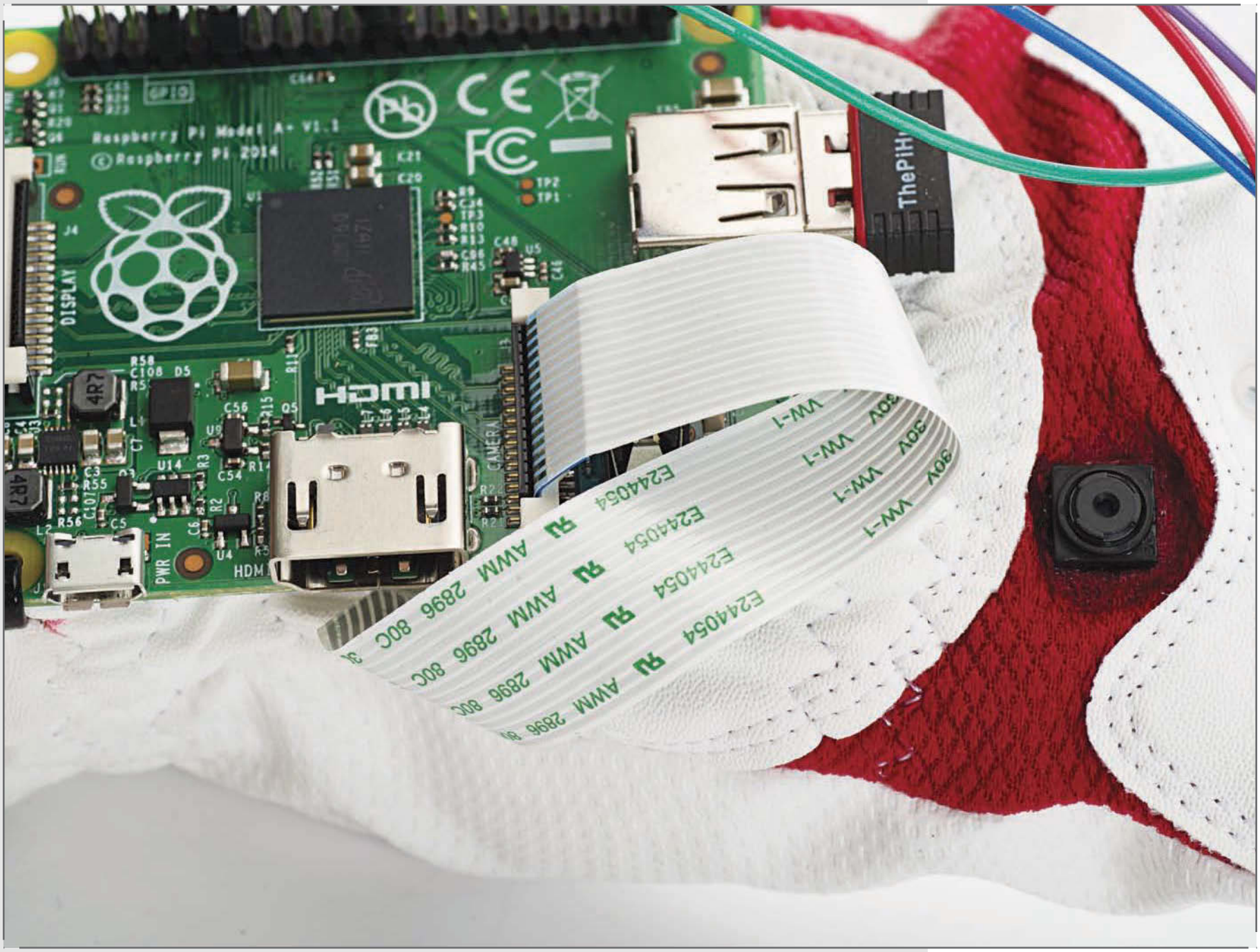
05 Attach the Pi camera cable

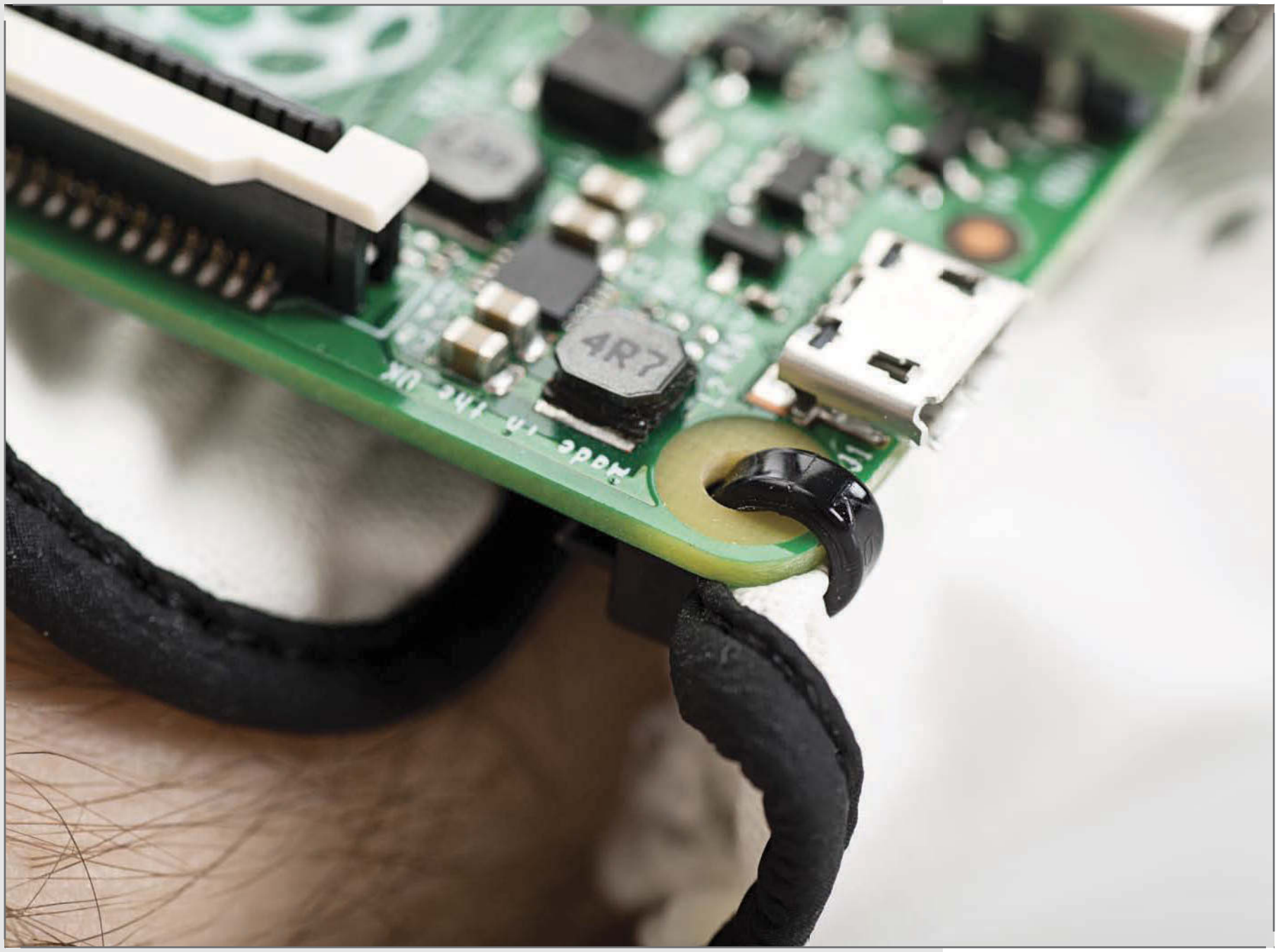
Flip the glove inside-out so you can access the back of the camera. Take the camera connection wire, with the blue side facing away from the camera, and attach the wire to the back of the camera. On the Raspberry Pi, slide up the black camera holder piece and secure the wire into place. You may wish to thread the wire through the side of the glove to conceal it.

06 Checkpoint

At this stage you have the five poppers attached individually to a wire and through each of the individual fingers, and also attached to the thumb. Each of these

Below We placed our camera on the back of the hand, but you can put yours anywhere you like





wires runs through the inside or outside of the glove to the Raspberry Pi. The Pi camera is also attached to the glove, with the lens exposed to enable pictures to be taken.

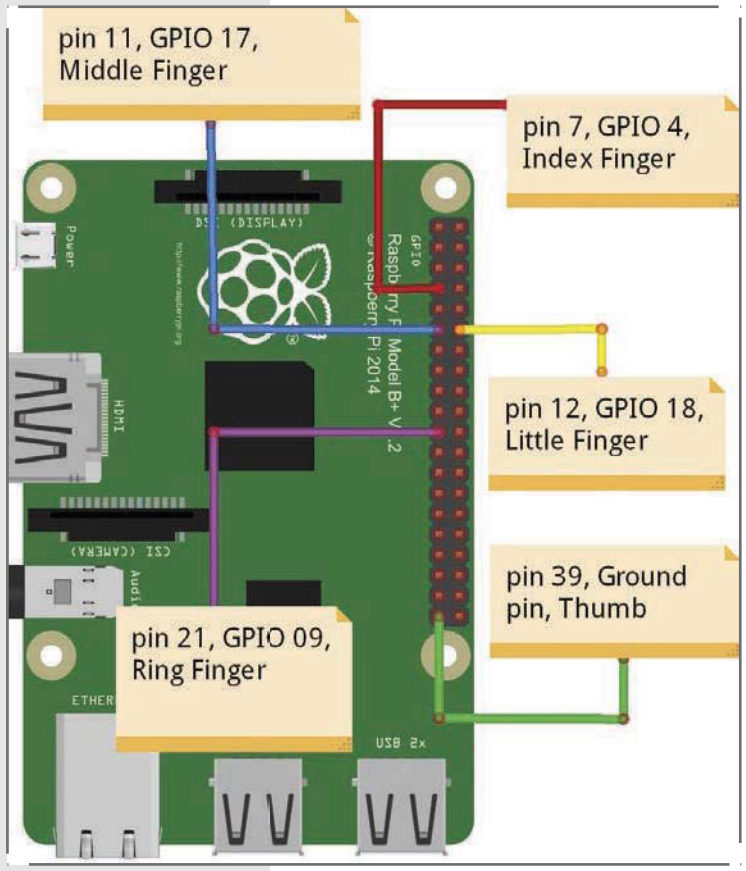
07 Mount the Raspberry Pi

You can mount the Raspberry Pi onto the top of the glove using the small cable ties and the four holes located on each corner of the Pi – this makes the overall glove more portable. Take your Pi and orientate it so that the SD card slot is facing away from your wrist. Thread the cable ties through the holes and through

the glove, securing the Pi in place. Be careful not to over-tighten the cable ties.

08 Attach the wires to the GPIO pins

Now that all the hardware is attached to the glove, the next step is to connect the finger and thumb wires to the GPIO pins on the Raspberry Pi. The pin numbers we use here identify the physical pin number on the board. Take the thumb wire and connect it to physical pin number 39 – this is the ground contact for the other buttons. Now take the index finger wire and connect to physical pin number 7.



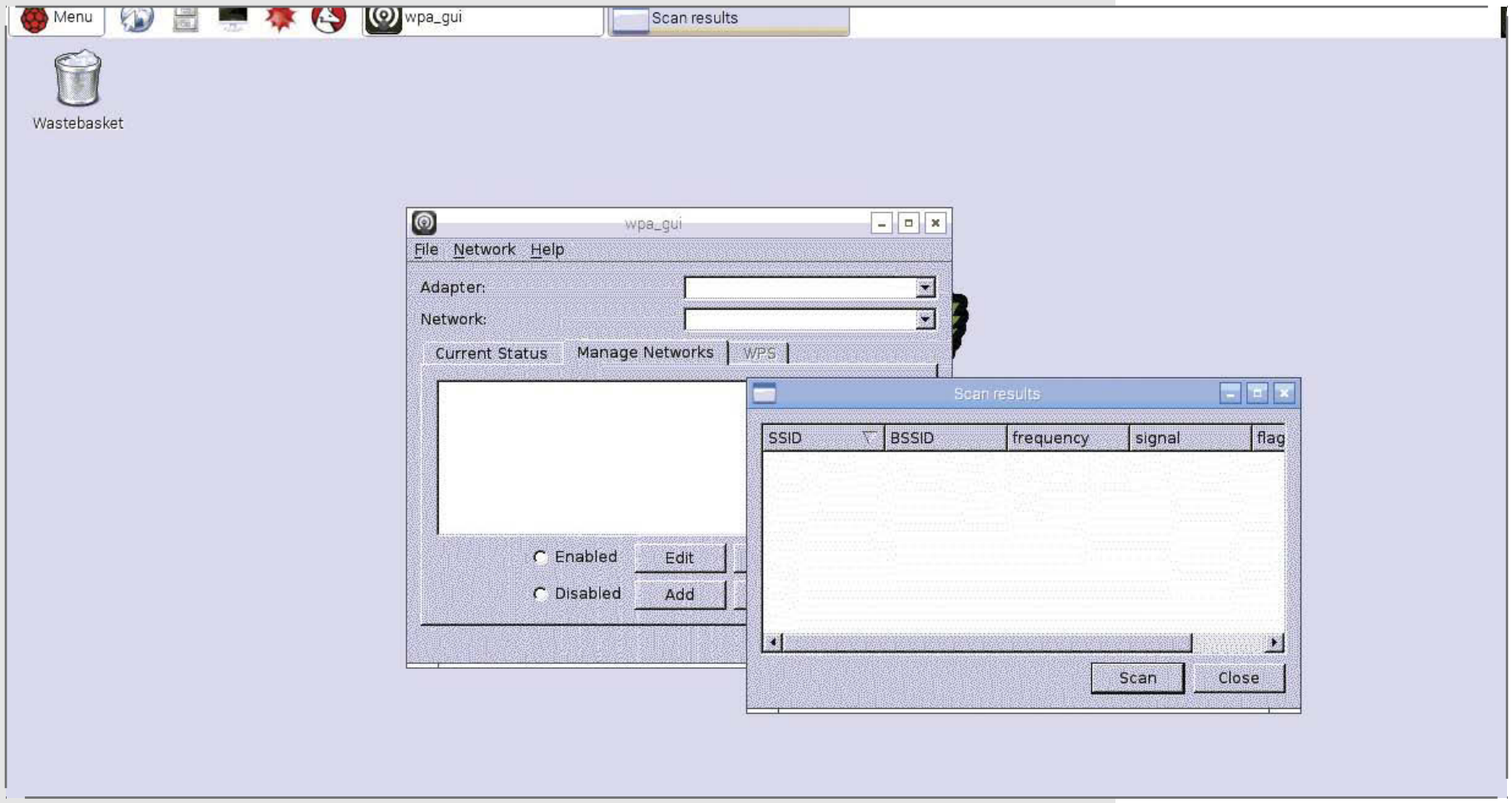
Above It won't fit onto your glove as neatly, but you can use the Model B+ or 2B as well as the A+

09 Connect more wires to the GPIO pins

Using the same method as described in Step 8, take the middle finger wire and connect it to pin number 11, the ring finger to pin 21 and finally the little finger wire to pin 12 – again, these numbers are the physical pin numbers on the board.

10 Boot up the Pi

The first time you boot up your Raspberry Pi you will need to set up your Wi-Fi connection. Connect the Pi to your HDMI monitor and insert the Wi-Fi dongle. If you are using the newer A+ model, which only has a single USB, you may require a USB hub to enable you to add a keyboard and mouse. You could always set up the Wi-Fi on a B+ model, which offers more ports, then transfer the card to the A+. Add the USB mouse, keyboard and power supply. Boot the Pi up.



11 Add Wi-Fi app

Open up the Wi-Fi Configuration tool from Menu> Preferences. When open, select Manage Networks and then Scan – this will find and list all of the available Wi-Fi networks in your area. Select your network from the list and double-click. You will then be prompted to enter your WEP code, so enter this in the PSK text box and press Save. Return to the Current Status page and click Connect. This will connect your Pi Glove to the Wi-Fi.

12 Set up Wi-Fi in the command line, part one

If you are using SSH to access your Pi, you can set up your Wi-Fi dongle from the command line. In the terminal window, type: **sudo iwlist wlan0 scan**. This will scan for available Wi-Fi networks. Look out for: ESSID:"test-network" – this is the name of the Wi-Fi network.'IE: IEEE 802.11i/WPA2 Version 1' is the authentication that is used; in this case it is WPA2,

the newer and more secure wireless standard which replaces WPA1. You will also need your password for your Wi-Fi network – for most home routers, this is located on a sticker on the back of the router. The ESSID (ssid) for the network in this case is 'test-network' and the password (psk) is 'testingPassword'.

13 Set up Wi-Fi in the command line, part two

Now add your Wi-Fi settings to the wpa-supplicant.conf configuration file. In the terminal window, type:

```
sudo nano etc/wpa_supplicant/
wpa_supplicant.conf
```

Scroll to the bottom of the file and add the following lines:

```
network={
    ssid="The_ESSID_from_earlier"
    psk="Your_wifi_password"
}
```

Using the example network found in Step 12, you type ssid="test-network" and psk="testingPassword". Now save the file by pressing Ctrl+X then Y, then press Enter.

14 Set up Wi-Fi in the command line, part three

On saving the file, wpa-supplicant will normally notice that a change has occurred and so it will try to make a connection to the network. If it doesn't do this, you can either manually restart the interface – just run sudo

"The Pi camera is mounted into the glove, which enables you to angle your wrist and take a picture"

ifdown wlan0 followed by sudo ifup wlan0 – or instead reboot your Raspberry Pi with sudo reboot. To test that the Pi is successfully connected to your Wi-Fi, type ifconfig wlan0. If the 'inet addr' field has an address beside it, the Pi has connected to the network. If not, check that your password and ESSID are correct.

15 Disable the Wi-Fi power management

If left idle, the Wi-Fi power management system on the Raspberry Pi may drop the Wi-Fi connection – this may, for example, occur if the glove is out of range of the router. To disable the power management, load the terminal window and type:

```
sudo /etc/network/interfaces
```

At the end of the block of code, add the following line:

```
wireless-power off
```

This will ensure that the Wi-Fi stays connected whilst in range.

16 A simple test program

Now you have completed the hardware section of the Pi Glove, you can use a simple program to test the connections and make sure that all of the poppers are working correctly and responding to the thumb and finger contacts. Download the test program from our site. Run sudo idle in a terminal to open the Python editor, then start a new file and insert the code. With your Pi Glove attached, save and run the program.

SSID

SSID is a case-sensitive, alphanumeric, 32-character unique identifier attached to the header of packets sent over a wireless local area network (WLAN) that acts as a password when a mobile device tries to connect to it.

17 Run the code

The test program will respond to each connection of the fingers and the thumb, and display a message stating that the respective button has been pressed – like so: 'button one has been pressed', 'button two has been pressed', etc. If this fails, check for the following errors: 1) incorrect wiring on the GPIO pins, 2) loose wires not in contact with the poppers, and 3) thumb and finger not in contact. Shortly we will cover how to develop a program that brings control to your fingertips.

18 A quick test and recap

Ensure that your glove hardware consists of at least five wires connected to a Pi which is mounted to the glove. A Pi camera is also embedded or attached to the glove. Boot up your Raspberry Pi; this could be a separate Pi. Initially, it is worth running the test program below, to ensure that all the hardware and wires are connected correctly and working properly.

```
import RPi.GPIO as GPIO
######Set up the GPIO Pins ######
GPIO.setmode(GPIO.BCM)
###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
##11 on the BOARD
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
##12 on the BOARD
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
##21 on the BOARD
```

"On saving the file, wpa-supplicant will normally notice that a change has occurred and try to make a connection to the network"

```
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
##7 on the BOARD
GPIO.setwarnings(False)  ##switch off other ports

while True:
    if GPIO.input(4) == 0:
        print “You pressed button one”

    if GPIO.input(17) == 0:
        print “You pressed button two”

    if GPIO.input(9) == 0:
        print “You pressed button three”

    if GPIO.input(18) == 0:
        print “You pressed button four”
```

19 Install the Python libraries

Assuming all went well with the test, you are set up and ready to build the new program. The good news is that most of the modules that you will use in the program are already pre-installed on the Raspbian operating system. To add the ‘spoken instructions’ feature you will install a module called eSpeak. In the LX Terminal, type:

```
sudo apt-get install espeak python-espeak
```

To play the MP3s, syou will use a program called mpg321. To install this, type:

Twitter

The original Project New York Glove featured the ability to tweet the picture that was taken with the Pi camera. If this is a feature you are interested in, you can read more here about how to set up a Twitter API (http://www.tecoed.co.uk/twitter-feed.html). Button 4 also retrieved the train times between two stations then stored and read these out through the headphones; further details can be found over here: http://www.tecoed.co.uk/scraping-trains.html

```
sudo apt-get install mpg321
```

Once installed, restart the Pi.

20 Test eSpeak

eSpeak is a simple, compact, open source software speech synthesiser that uses English and other languages. It works by taking a text string and then converting it into audio. But that's not all – you can also edit the voice, pitch, volume and other parameters of the speech. Test that it is working by creating a new Python file and using the code **espeak.synth** ("This is a Test"). Now, when you run the program it will read out the phrase "This is a test".

```
from espeak import espeak
espeak.synth ("This is a test")
```

21 Import modules

The Glove program uses a number of modules for its various functions. These are Python files that contain a number of definitions, variables, functions and classes. Import the modules below into your Python program – these will give your program access to the MP3 player, the Pi camera module, the GPIO pins and eSpeak.

```
import time
import random
import os
import sys
import subprocess
```

"You can now create your own interactions for your glove – for example, turn lights on and off or send an SMS to a mobile phone"

```
import picamera
import RPi.GPIO as GPIO
from sys import exit
from espeak import espeak
```

22 GPIO pull-ups

To register that you have triggered the finger button, we make use of GPIO pull-ups to detect that the two contacts have touched together. The pull-up resistor sets the current to 0 volts. When the two wires connect, the voltage changes and this change in state is recognised, triggering the function which you will assign to each of the buttons. If you have no pull-up or pull-down then the GPIO pin can change state, for instance if there is external interference, and this means that it can trigger your button even if you did not touch it. To set these up, add the following code to your program:

```
GPIO.setmode(GPIO.BCM)

###sets the pin to high ###
GPIO.cleanup()
GPIO.setup(17, GPIO.IN, GPIO.PUD_UP)
##11 on the BOARD SPARE
GPIO.setup(18, GPIO.IN, GPIO.PUD_UP)
##12 on the BOARD MUSIC PLAYER
GPIO.setup(9, GPIO.IN, GPIO.PUD_UP)
##21 on the BOARD TAKE A PICTURE
GPIO.setup(4, GPIO.IN, GPIO.PUD_UP)
##7 on the BOARD TIME
GPIO.setwarnings(False)##switch off other ports
```

23 Add the spoken instructions

Since there is no visual display, you will not know that the program is running or that it is ready. Therefore, at the start of the program it reads out the button number and the function of each. This uses the same code from Step 20, calling eSpeak to convert the text to an audio wave and play it back through a speaker or a pair of headphones. You can customise the introduction and what instructions are given. Use **time.sleep(2)** to add a slight break between the sentences and make the speech more natural.

```
espeak.synth ("Welcome to the PI GLOVE")
time.sleep(2)
espeak.synth ("Please make a selection")
time.sleep(2)
espeak.synth ("Button 1 - tell you the time")
time.sleep(2)
espeak.synth ("Button 2 - take a picture")
time.sleep(2)
espeak.synth ("Button 3 - play some tunes")
time.sleep(3)
espeak.synth ("Please select your button")
```

24 Set up the time

At this point you are now ready to set up the function for the first button, which will tell you the time in a similar fashion to the old 'speaking clock'. This feature means you don't have to take out and unlock your phone – simply press the button and the current time is read back to you. Line 2 of the code creates and stores the current time as a variable

OS in Python

The Python OS module enables you to interface with an operating system, which provides a way to use Python to interact with a Linux, Windows or Mac computer. Python code can then be used to control OS system commands such as changing file names, creating folders and files, as well as changing file paths. You can also find out information about your location or about the process.

```
current_time = (time.strftime("%H:%M:%S"))
```

A second variable, line 3, is used to store the 'time message' which is then used by eSpeak to read out the time to you, line 4. Add the code to your program:

```
def what_is_the_time():
    #global time
    current_time = (time.strftime("%H:%M:%S"))
    the_time = "The current time is %s" %
current_time
    espeak.synth(the_time)
    time.sleep(2)
```

25 Set up the camera

The picamera module is pre-installed on the Raspberry Pi, so you are ready to create a function which will trigger the camera and save the picture as a new file called newpic.jpg (line 5). The third line is useful to test that the camera is taking a picture and also to familiarise yourself with where you are pointing the camera. When triggered it will display a preview of what the camera sees on a connected monitor or television.

```
def take_a_pic():
    with picamera.PiCamera() as camera:
        camera.start_preview()
        time.sleep(2)
        camera.capture("newpic.jpg")
```

26 Take a selfie

It is possible to perform a test in order to ensure that the camera is working by calling upon the take_a_pic() function. Do this by opening up a new Python window and then add the previous code from Step 8. Save and run the code, and you should see a two-second preview of what the camera sees and then the camera will capture this image, which is then stored in the Pi/Home folder.

27 Save as a new file name

Currently, each time a new picture is taken, it overwrites the previous file. Annoyingly, this means that you will lose the last picture you took. To stop this, create a global variable called File_Number, line 1. This variable is incremented each time a new picture is taken. Create

a second variable, called file_name (line 2) – this variable is combined with File_Number to create a new unique file name each time the picture is saved (line 4), preserving your previous pictures. Line 5 ensures that the File_Number value is incremented by one each time a photo is saved.

```
global File_Number ###number if photo
global file_name ###name of photo
File_Number = 1

file_name = “Picture” + str(File_Number) + “.jpg”
            File_Number = File_Number + 1
```

28 Final camera code

The complete camera code uses a function that combines the features from the previous steps to trigger the camera and save the image as a new file with a unique file name each time the two ‘poppers’ connect. Add the code below to a new line underneath your time function.

```
def take_a_pice(): ###Takes a picture ###
    global File_Number
    global file_name
    with picamera.PiCamera() as camera:
            time.sleep(0.5)
file_name = “Picture” + str(File_Number) + “.jpg”
             File_Number = File_Number + 1
```

29 Save the music

There are two small steps to take before you can enable the music player. First, download a number

of MP3s and save the file names as numbers – for example, 1.mp3, 2.mp3, 3.mp3 and so on. For the second step, create a variable at the beginning of your program to store the file names, such as:

```
songs_list = [“1”, “2”, “3”, “4”, “5”]
```

This variable is used to select the song.

```
###Code for MP3 random play list###
songs_list = [“1”, “2”, “3”, “4”, “5”]
```

30 The music player

Instead of creating another function for the music playback, the MP3 player is called directly from the GPIO pin 17 pull-up. It makes use of the variable song_list, which holds the file names stored as a list of numbers: 0,1,2,3,4,5. In the Home folder, you'll have your six music tracks named 0.mp3, 1.mp3, 2.mp3, etc. In order to make this a shuffle-based player, we can use the following line of code:

```
os.system(‘mpg321 ’+ (random.choice(songs_list))
    + ‘.mp3 &’)
##change
```

… which calls the operating system to load the mpg321 software and select a random number from the play list, and it then loads and plays the corresponding mp3 file.

31 Stop the music

The code in in this step will keep the music playing continuously. To stop the music, use the code:

```
os.system('sudo killall mpg321')
```

Map this code to the button on the glove and, by holding down the button for a little longer, you can cycle through a variable called song_play (line 3), which changes from 'yes' to 'no'. When the variable is 'no' (line 9), a conditional is used on line 4 to check the state and then use the following code:

```
os.system('sudo killall mpg321')
```

… in order to stop the music playing (line 8). Listen to the spoken instructions, and you can then time it right to properly end the music. Now we've explained what's going on, add the following code into your program:

```
os.system('sudo killall mpg321')
        espeak.synth ("Music Player ")
        song_play = "yes"
        if song_play == "yes":
            os.system('mpg321 '+ (random.
    choice(songs_list)) + '.mp3 &')
            if GPIO.input(17) == 0:
             #turns off song longer hold
                os.system('sudo killall
    mpg321')
                song_play = "no"
                espeak.synth ("MP3 player
    stopped")
```

32 Create the button structure

Now you have created three features for your glove, you can start assigning them to the finger buttons, which will trigger each function when they are connected with the thumb button. This is achieved with a simple set of conditionals, like: if GPIO.input(17) == 0:. This checks if a GPIO pull-up is present, and then if so, it runs the assigned function. Add the four conditionals below into your program. Remember to move the music player code so that it's beneath the GPIO pin 17 code.

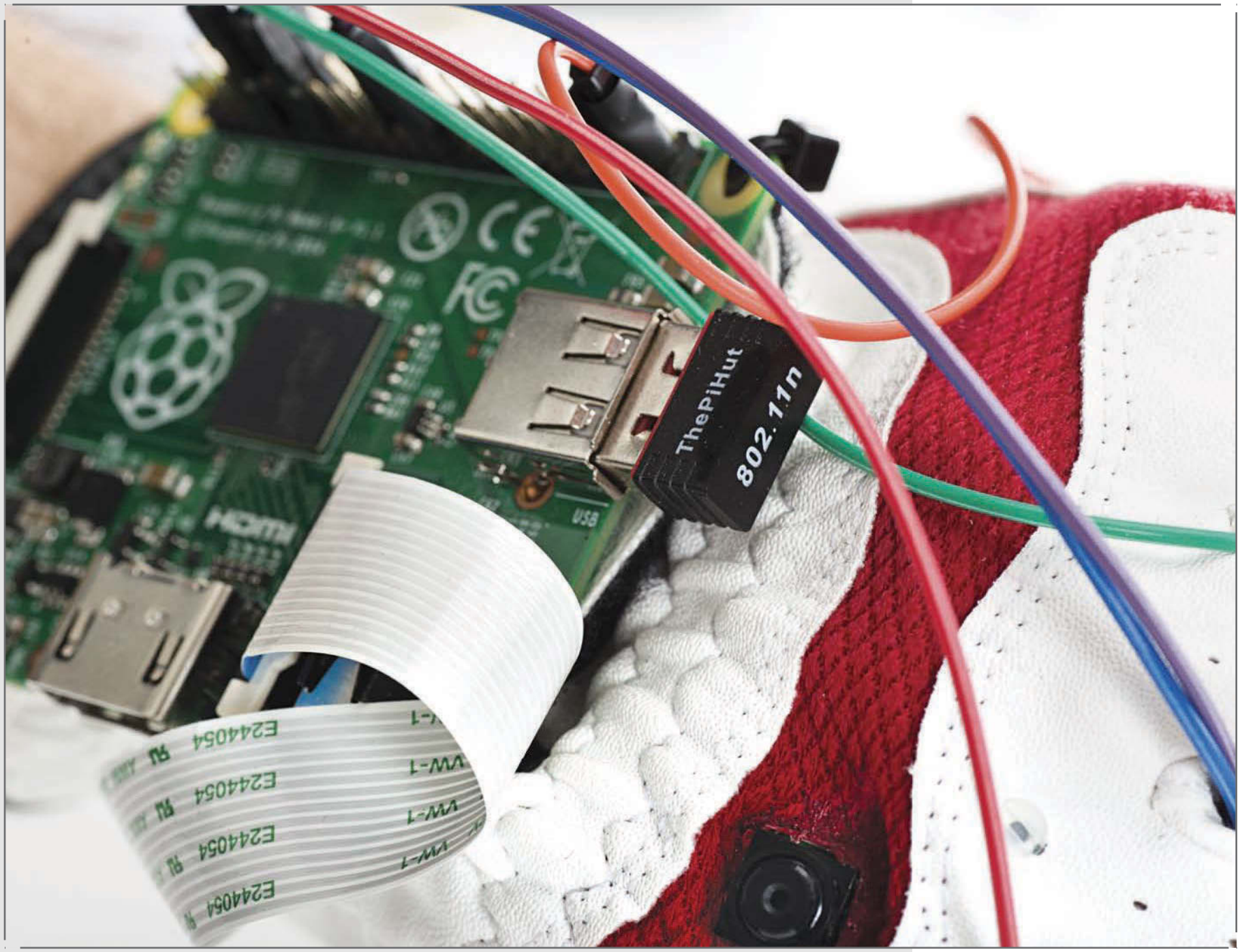
```
while True:
    if GPIO.input(4) == 0:
    if GPIO.input(9) == 0:
    if GPIO.input(17) == 0:
    if GPIO.input(18) == 0:
```

33 Call the functions

Once you have created your conditionals to test for the button pull-up, the final part of your program is to add the function for each GPIO. For example, add the time function to GPIO pin 4 with:

```
if GPIO.input(4) == 0:
        what_is_the_time()
```

This will run the time function you created each time the pin 4 button is connected to the thumb button. You will also want to add some instructions under each button to inform the user what is happening. For example, when triggering the camera it



is useful to know when the picture is going to be taken – have a look at the code example below.

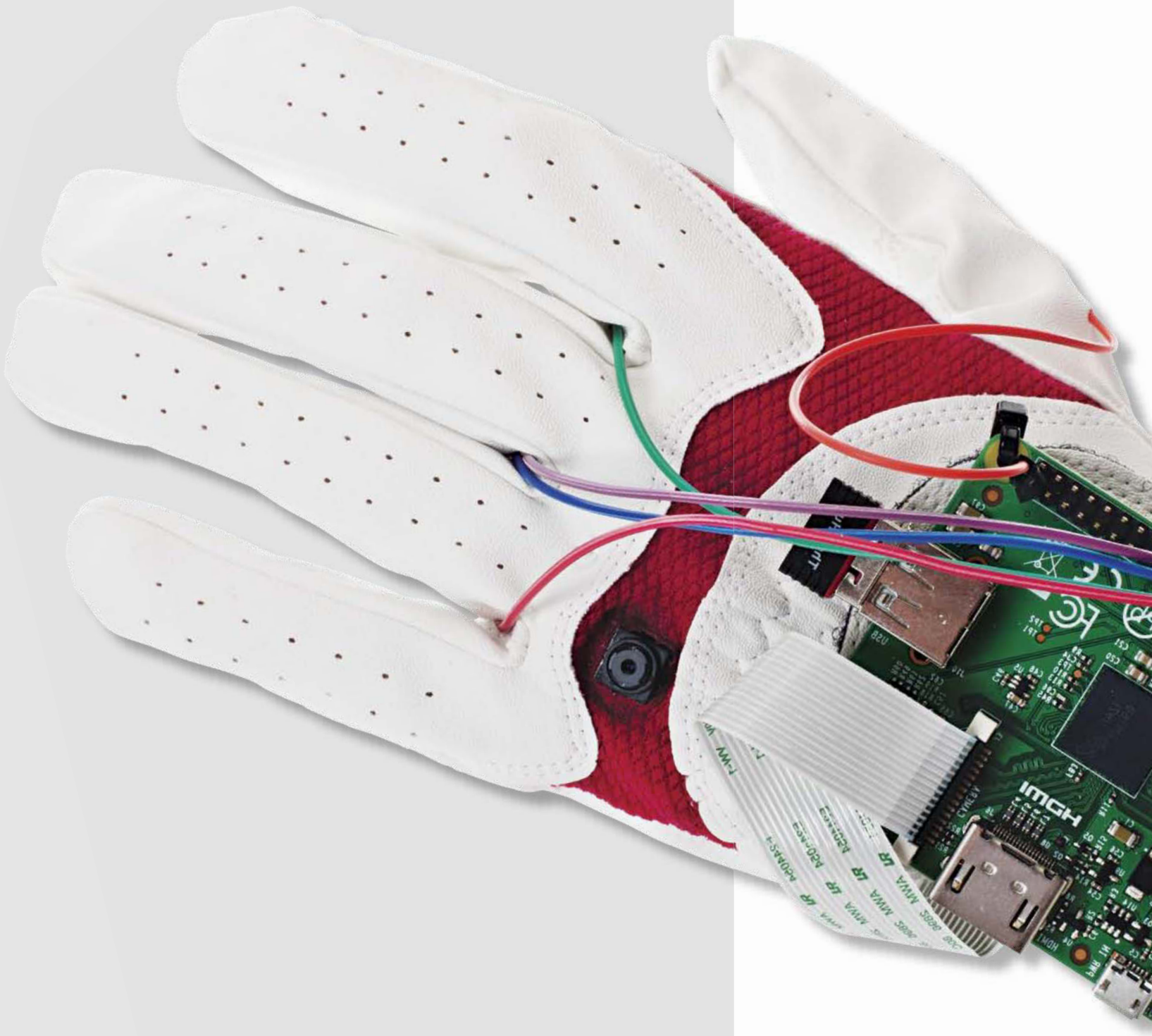
```
time.sleep(1)
espeak.synth("Preparing the camera")
time.sleep(2)
espeak.synth("Camera ready, smile")
time.sleep(1)
take_a_pic()
###enables the camera def and takes a picture
espeak.synth("Picture taken and saved")
time.sleep(3)
```

Above We're using the brilliant Wi-Fi dongle from The Pi Hut, which you can get for just £6: http://bit.ly/1LfkCgZ

```
espeak.synth("Press button two to tweet your
picture")
```

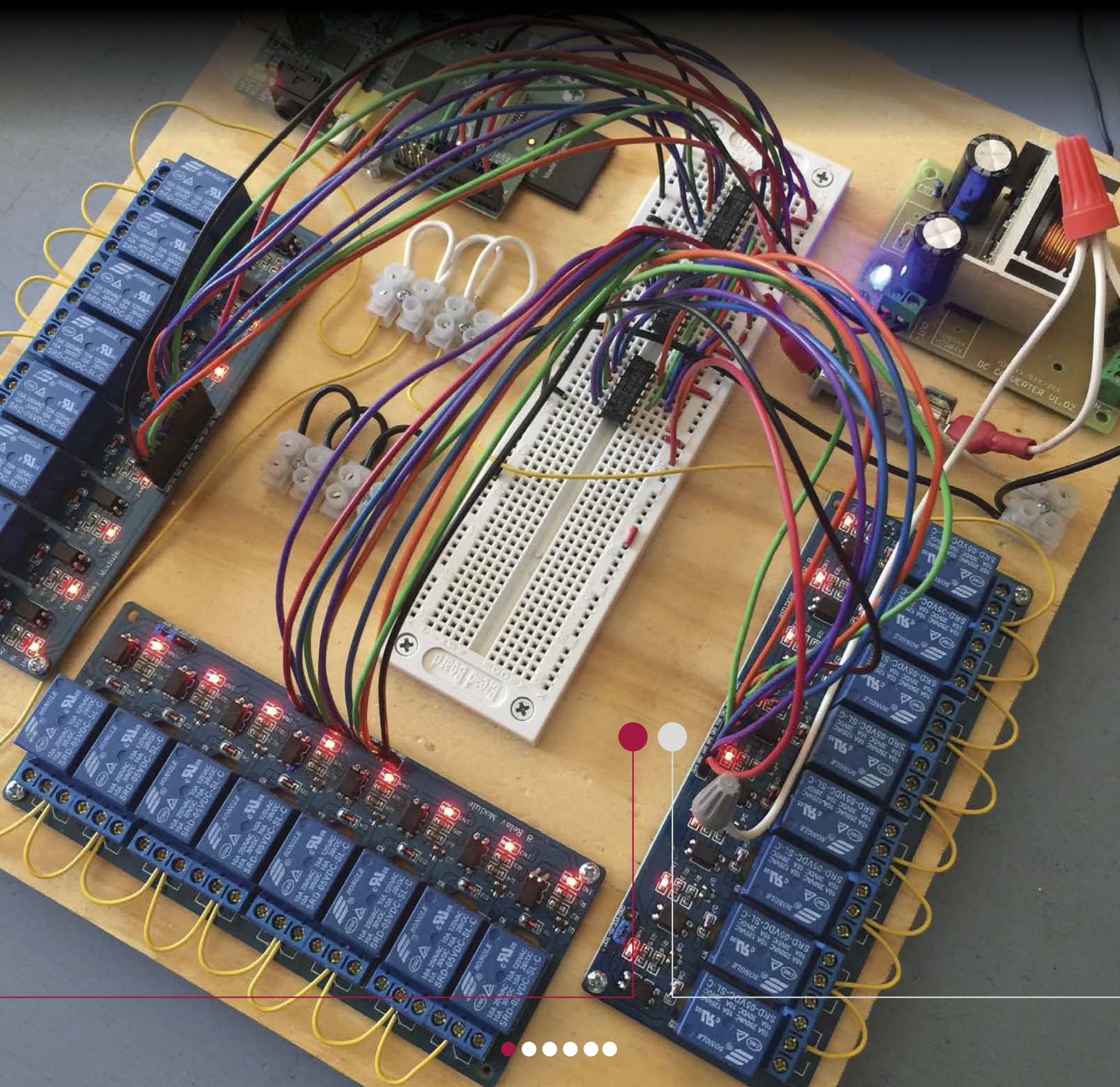
34 Other functionality

Save and run the program as the root user, then debug the code and test the contacts. Common errors may be incorrect wiring on the GPIO pins, loose wires not in contact with the metal poppers, or the thumb and finger not in contact. Once working, you can now create your own interactions for your glove – for example, turn lights on and off, send an SMS to a mobile phone, control your TV or read text in a text file.



Fireworks Controller

Chris Osborn shows how your Raspberry Pi can light up the Fourth of July



How did you first get started with this project?

So, I'm building a sprinkler controller and I realised that, before I installed it as a sprinkler controller, I could use it for fireworks. It's actually installed as a sprinkler controller now.

What hardware did you decide to use to build it?

I just got a bunch of 8-channel relay boards from eBay and some power supplies that would take up to 48 volts in, and do a DC-DC step-down (but it will actually take AC in as well, because that's what I need for the sprinkler controller), and it was like $4 for the power supply and I think $7 a piece for the relay boards. I connected three relay boards for the sprinklers, but I actually only used one relay board to control the fireworks. So yeah, there's pretty much nothing to this. The only other thing I had to use was a shift register, a 74-595, to control the relay board. I don't think there's any other chips on it, and I did have to use a resistor for a pull-up to keep the relay board disabled until the software starts up, but otherwise there's nothing to it.

Walk us through the circuit – what exactly is going on?

The shift register is connected to the GPIO on the Pi and I think that uses four pins – Enable, Clock, Data and Latch – and so it just shifts the data out serially, one bit at a time, and since the 595 control has eight bits and I have eight relays on a single board, one shift register for one board, I just shift out which relays I want to turn on and then latch it in, and they turn on. The only thing that was a little bit tricky, which for the fireworks was not an issue, is that the relay board is Active Low rather

Chris Osborn is into programming, coin-op, woodworking, electronics and retro technology, often combining skills learned from all of those things in order to make unique or unusual creations

THE PROJECT ESSENTIALS

Raspberry Pi B

AC/DC 9-48V to 1.8-25V 3 switching power supply

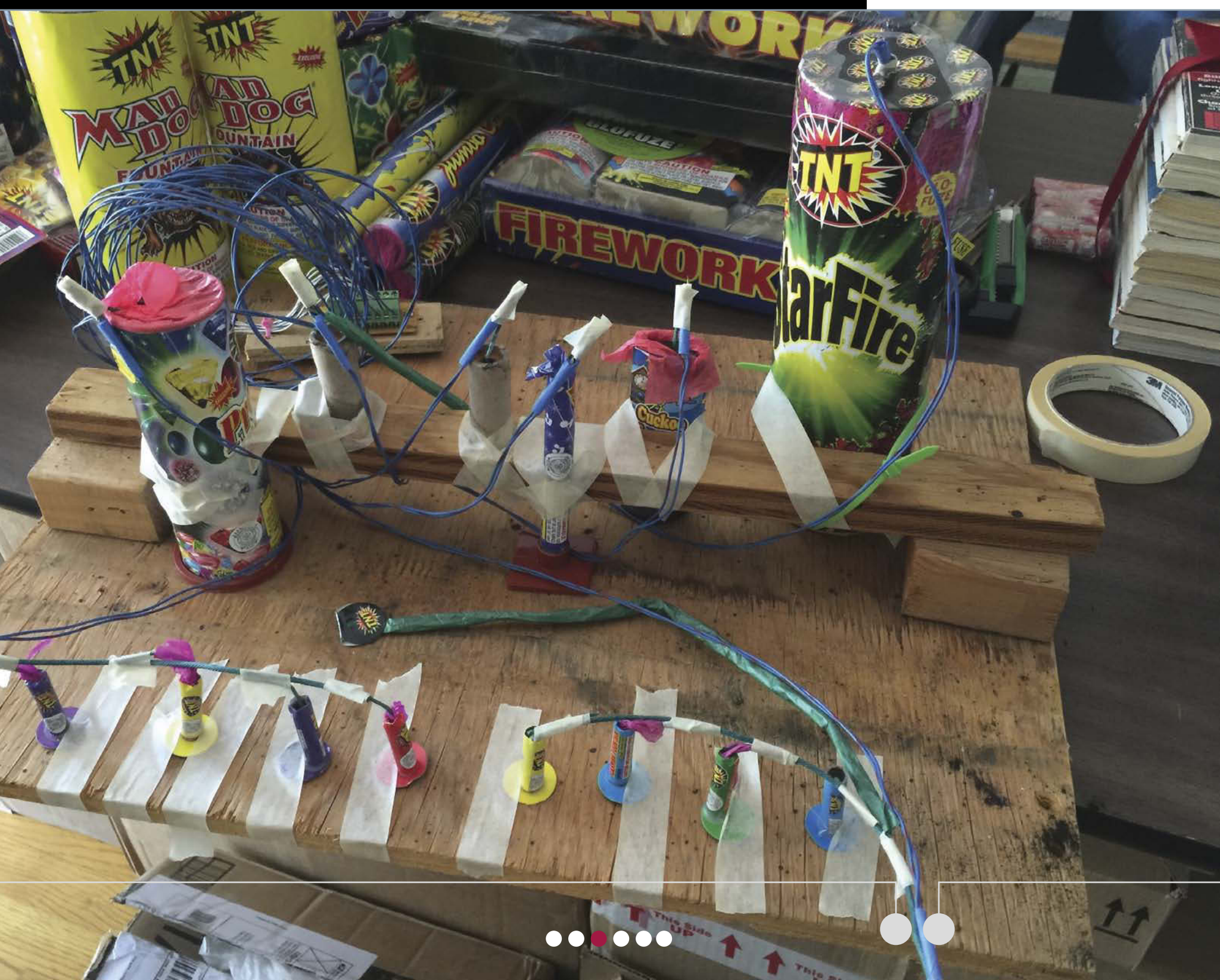
5V 8-channel relay module

74HC595 shift register

than Active High, and that seems to be pretty common with all of those relay boards that are like that. And the relay board that I used, I got one that uses 5-volt coils so I didn't have to feed it with any other voltages; so the five volts that feeds into the Raspberry Pi is also the same power supply that powers the relay board.

How exactly is the Raspberry Pi controlling the relay boards?

It's just some really simple software that I reused from a long, long time ago, when I was working with a much more complicated setup, and I just say which relay I want to turn on and then it just shifts out the data and



Below The fireworks were connected to the relay module and then lit using Chris' homebrew e-matches

turns it on. I put a time delay on it this time around so that after five seconds, it automatically turns the relay off.

Did you reuse the web interface from an old project as well, or was it custom-made for this?

Yeah, that's what I did; I wrote a Python script that runs… well, because of the way the GPIO works on the Pi, any script that wants to talk to the GPIO has to be root, so I just made that into a separate little program and I just tell it which relays to turn on. And then the web [interface] just calls that, and I had to do a sudo on that so I actually opened up a slight security hole just so Apache could do a sudo to call the script that enables the relays, but, you know, the Raspberry Pi was doing that for maybe ten minutes in my back yard, where somebody would have to be standing right next to it with their phone to hack into it. But yeah, the web interface was just an old, old, old C program that I had lying around, and it's kinda clunky, but it was from the previous setup that I had where I was using a laptop, access point and the gigantic relay board, and all that stuff was hooked up together and it was just such a mess that the last time it didn't go right, I decided to not do anything with that for several years. So I just reused that because it was way faster to get going with – all I had to do was change it so it called the Python script rather than the old, separate C program that it was using in order to control the GPIO.

Can you tell us what is was that changed your mind, then – why did you revive the fireworks project?

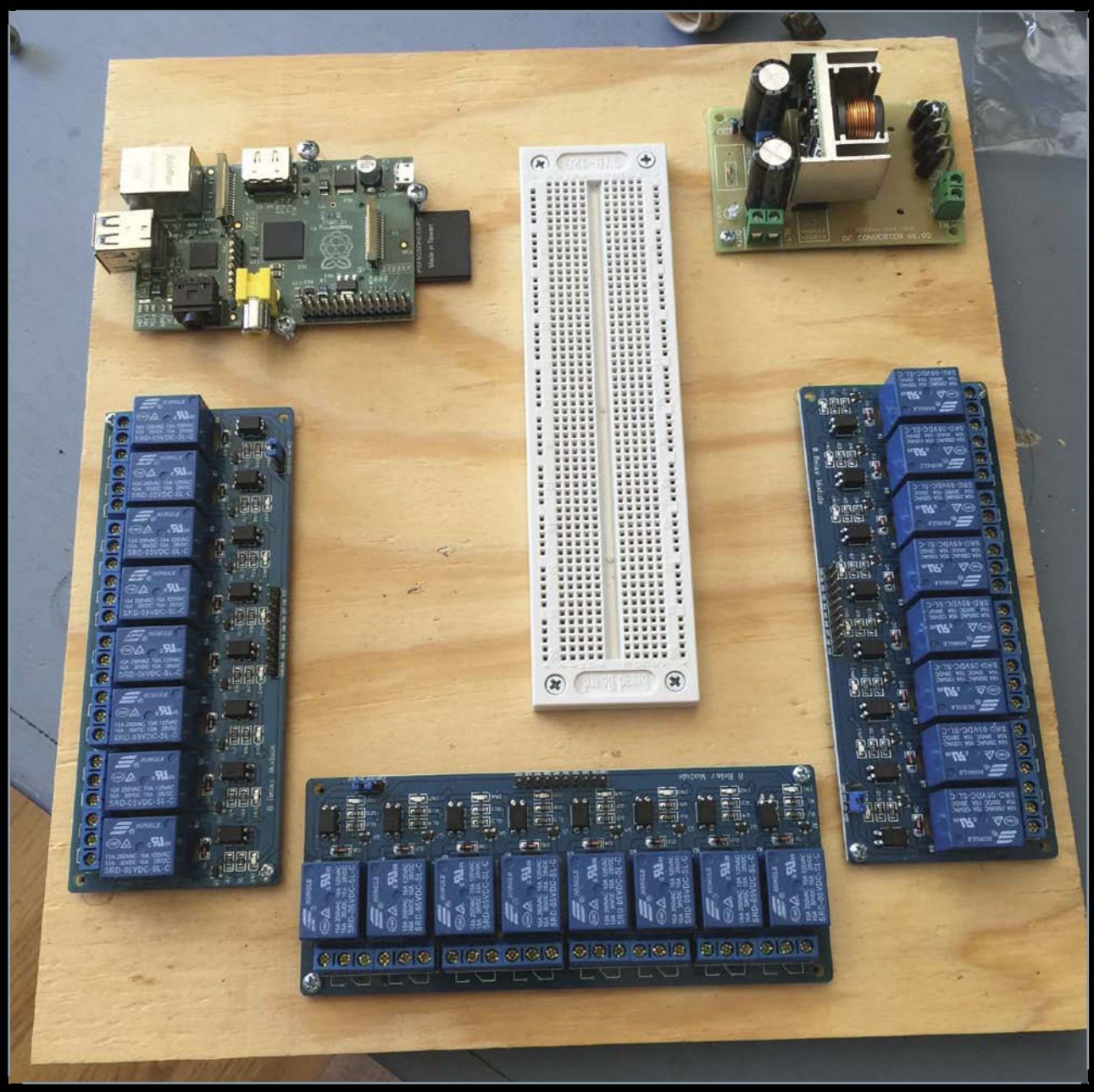
Because I've been thinking about it since the last time,

OS in Python

The Python OS module enables you to interface with an operating system, which provides a way to use Python to interact with a Linux, Windows or Mac computer. Python code can then be used to control OS system commands such as changing file names, creating folders and files, as well as changing file paths. You can also find out information about your location or about the process.

thinking that if I could simplify everything so it was all just one thing that I needed to bring out instead of four or five different things that I have to hook up, and so like I said I was planning to do the sprinkler project and I was planning to do it after the 4th of July, and then I realised, 'Wait a minute… these are just relays. If I have this assembled in time I could probably use it for fireworks.' I went ahead and put the whole together, and everything was on one board except for the battery.

This meant that the battery was the only thing that was separate, and I used a 12-volt sealed lead acid battery and fed that into the power supply, so all I had to do was bring down one board, one battery, and then



Like it?

If you fancy putting one of these together yourself, you can download Chris' circuit diagram (**http://bit.ly/1gq89uC**) and also his script for controlling the relays (**http://bit.ly/1Jy1f1h**), then just order the components listed on the other page.

Left For ease when heading to the show, and when installing as a sprinkler controller, Chris mounted the components on one board

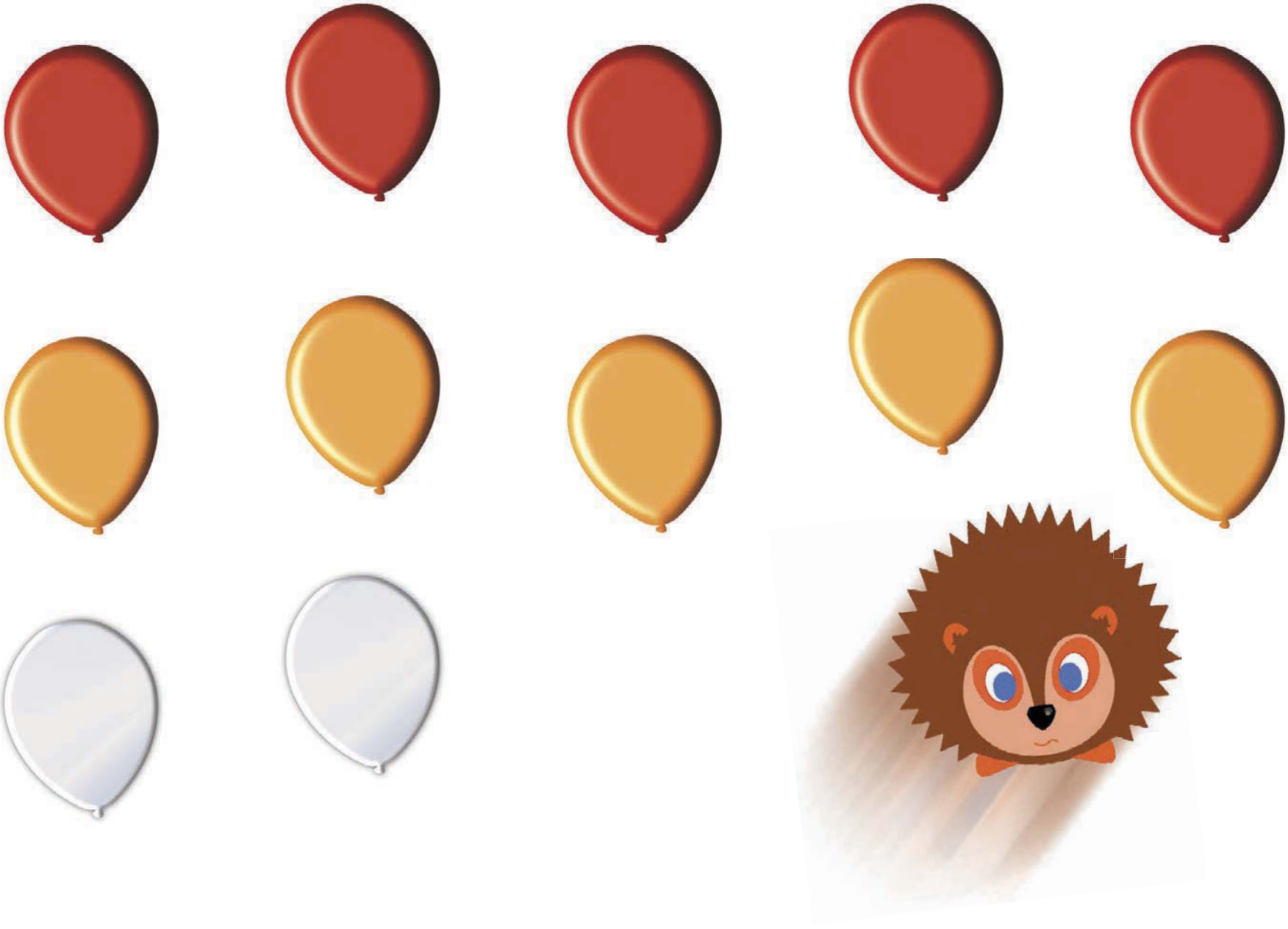
jumper-cable those together. So it was so much easier and I figured it was worth trying again, and I would say it was definitely much easier to set up. It didn't work perfectly – there were some problems where several of the fireworks didn't light, and I'm not exactly sure why because I actually saw the e-match glow, so I know that part lit and I'm not sure why the fuse didn't light. So I'm thinking that next time around I'll probably put some resistors on the relay switches so that when the relays are activated, the nichrome won't get maximum current but limited current, which means that instead of vaporising instantly it'll stay hot for longer.

Further reading

Chris uses his own e-matches to ignite the fireworks, essentially made from nichrome wire and paper matches. He's written a guide to making them (**http://bit.ly/1fLjzrT**) and you can watch the whole setup in action on YouTube (**http://bit.ly/1ODJAVr**).

Learn to code with FUZE BASIC

FUZE BASIC is a great first language to start learning how to program – here we will take you through the creation of a simple game, from start to finish



BASIC, prolific during the late Seventies and Eighties due to the popularity of the 8-bit BBC Micro, was the language that kickstarted much of the software industry we know today. Many programmers then moved on to more complex and powerful languages like C/+/++/Java etc, games consoles took over the home computer market and BASIC was all but forgotten.

Fast-forward 30 years and it's easy to see why the UK government is desperately trying to get kids coding again – resources are now very thin on the ground and we're outsourcing our programming requirements like there's no tomorrow. There's really never been a better time to become a programmer. You'll find no better introduction than learning to program a game, so we'll start with the classic bat-and-ball genre, but with a twist or two, of course.

To get started you will need to install FUZE BASIC and download the graphics required for the game from https://www.fuze.co.uk/lair and https://www.fuze.co.uk/FUZEBIN/spike.zip.

THE PROJECT ESSENTIALS

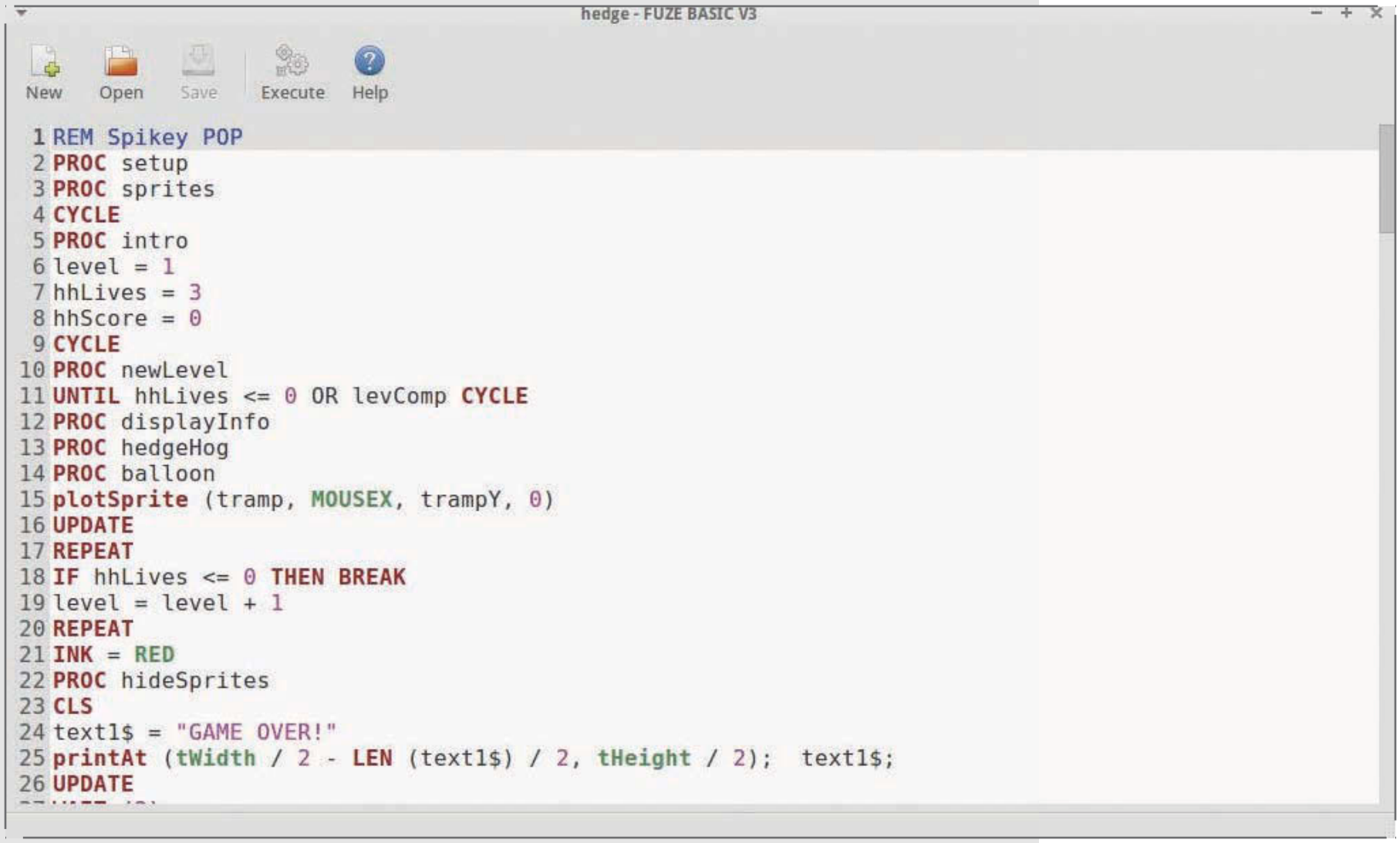
FUZE BASIC V3
https://www.fuze.co.uk/lair

Graphics
https://www.fuze.co.uk/FUZEBIN/spike.zip

01 Get started

After downloading and starting up FUZE BASIC, press F2 or type EDIT to get to the FB editor and then type in the following code. Capitalisation for any black text is critical.

```
REM Spikey POP
PROC setup
PROC sprites
CYCLE
PROC intro
```




```
level = 1
hhLives = 3
hhScore = 0
CYCLE
PROC newLevel
PROC getready
UNTIL hhLives <= 0 OR levComp CYCLE
PROC displayInfo
PROC hedgeHog
PROC balloon
plotSprite (tramp, MOUSEX, trampY, 0)
UPDATE
REPEAT
IF hhLives <= 0 THEN BREAK
level = level + 1
REPEAT
INK = Red
```

```
PROC hideSprites
CLS
text1$ = "GAME OVER!"
printAt (tWidth / 2 - LEN (text1$) / 2, tHeight
    / 2);  text1$;
UPDATE
WAIT (2)
CLS
REPEAT
END
```

02 Reset a new level

Next we need to enter the variables that we need to reset at the start of each new level:

```
DEF PROC newLevel
bals = maxB
metalCt = level - 1
IF metalCt >= 5 THEN metalCt = 5
PROC newLifeVariables
PROC hideSprites
plotImage (back1, 0, -20)
levComp = 0
PROC setupBals
ENDPROC
```

03 Reset variables

Now enter the variables to reset every time a life is lost, or at the beginning of a new level.

```
DEF PROC newLifeVariables
hhX = gWidth / 2
hhY = gHeight / 2.8
```

Game over?

Here we call the routines to set up the main graphics and variables. CYCLE starts the main game loop. The code then checks to see if you've completed the level or run out of lives – if that's the case, it will end the level.

```
hhYspd = .1
hhAngle = 0
hXdiff = 0
hXpow = 0
hYpow = .51
hhGrv = 0
hhXdirection = .1
hhYdir = 0
trampX = gWidth / 2
ENDPROC
```

04 Check the hedgehog

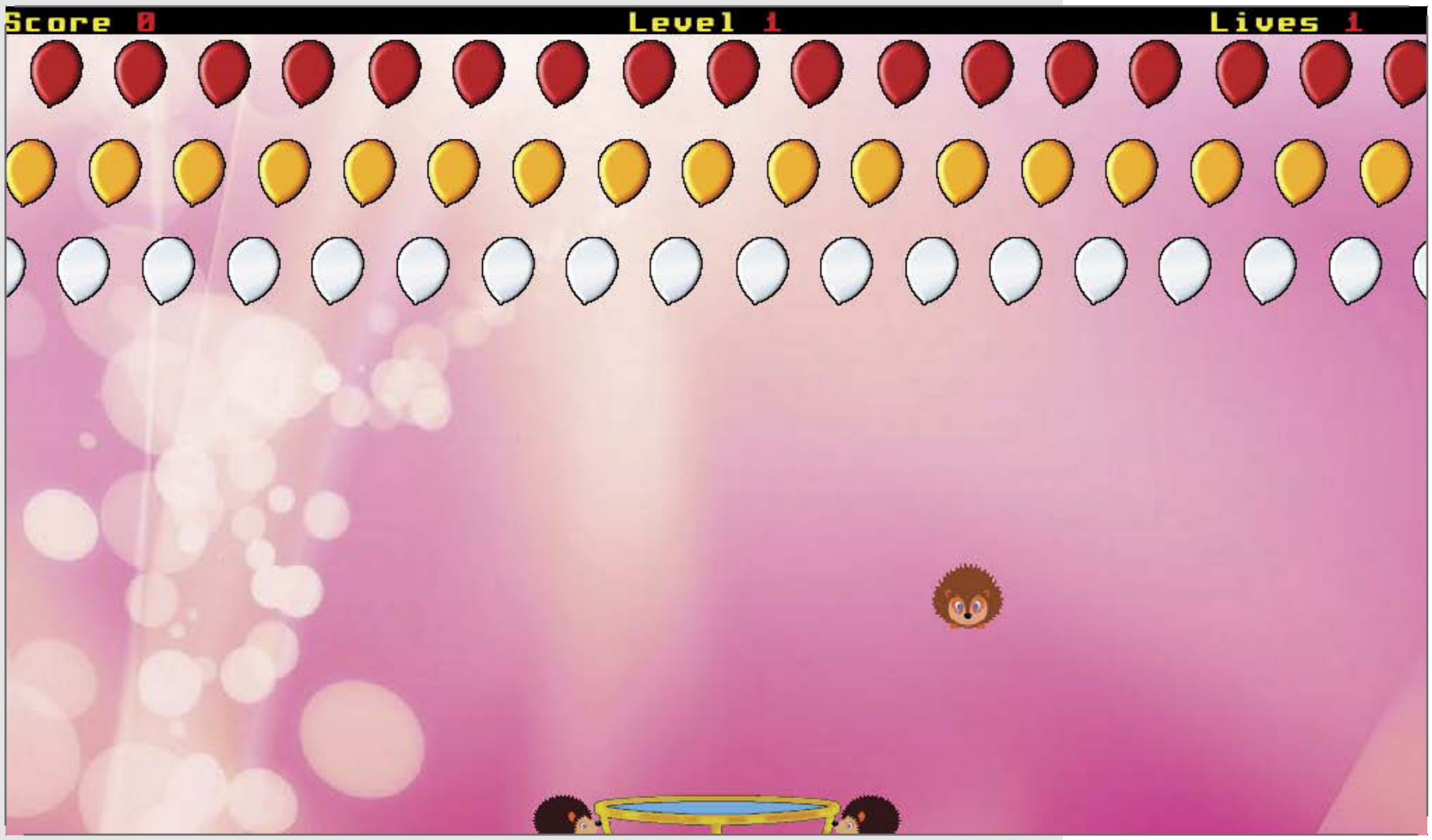
This section is where the hedgehog action is at. Here we check the position, the size and if Spikey has hit anything.

```
DEF PROC hedgeHog
hhGrv = (((gHeight - hhY / hYpow) / 80))
hhW = getSpriteW (hhID)
hhH = getSpriteH (hhID)
hCol = spriteCollidePP (hhID, 1)
IF hCol >= b(0, 0) AND hCol <= b(60, 0) THEN
IF  NOT b(hCol, 8) THEN
IF ABS (b(hCol, 2) - hhX) > 20 THEN
hXpow = (b(hCol, 2) - hhX) / 100 * RND (10)
hXpow =  - hXpow
ENDIF
```

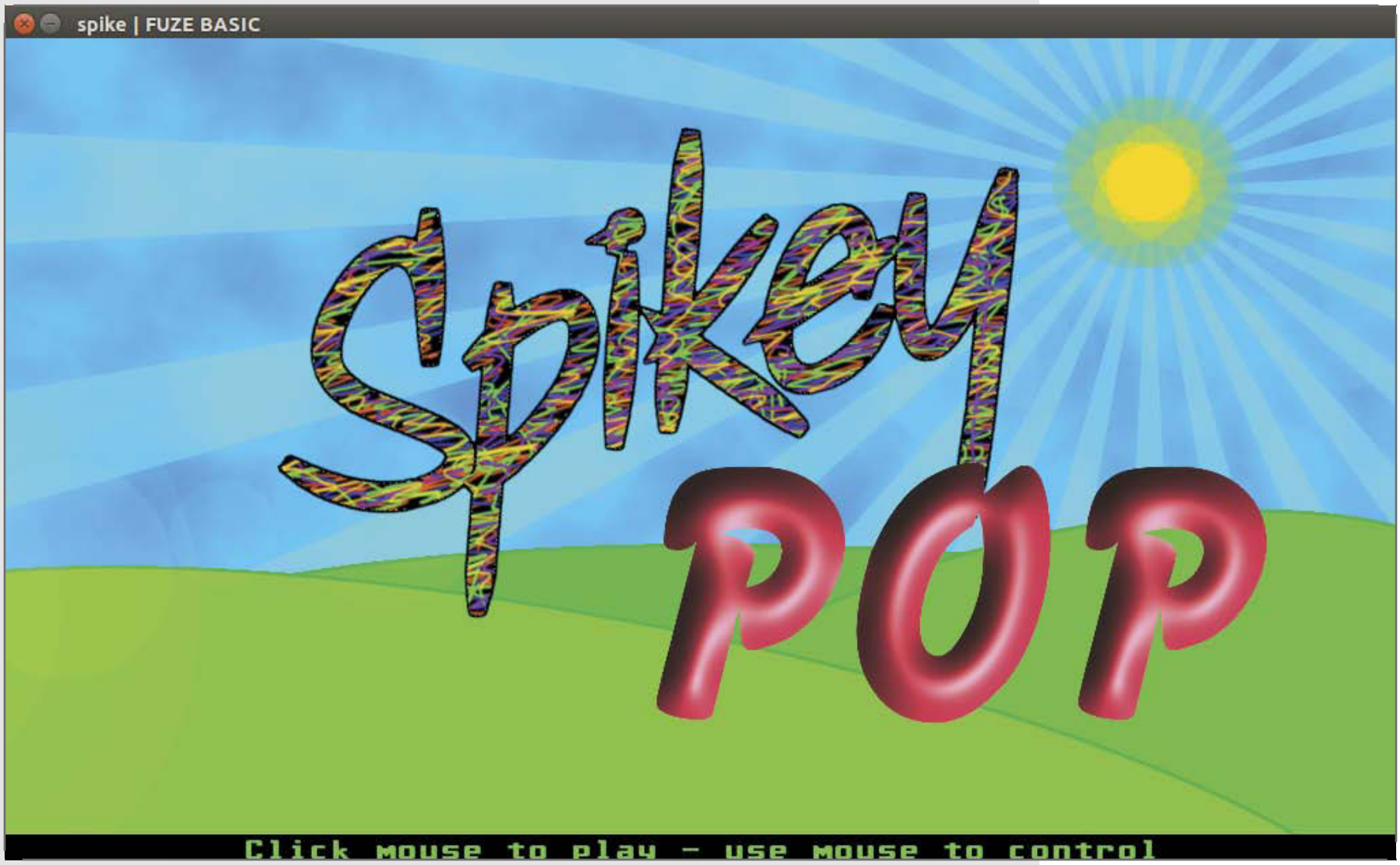
05 Check the balloons

Now we check to see which row of balloons has been popped. We also make a small speed adjustment so each time a balloon is hit the speed slowly increases.

"We'll start with the classic bat-and-ball genre, but with a twist or two, of course"




```
hhYdir =  NOT hhYdir
b(hCol, 8) = 1
IF hCol >= 0 AND hCol <= 19 THEN
hhYspd = hhYspd + 0.03 + level / 100
hhScore = hhScore + 200
ENDIF
IF hCol >= 20 AND hCol <= 39 THEN
hhYspd = hhYspd + 0.01 + level / 200
hhScore = hhScore + 100
ENDIF
IF hCol >= 40 AND hCol <= 59 THEN
hhYspd = hhYspd + 0.005
hhScore = hhScore + 50
ENDIF
bals = bals - 1
IF bals <= 0 THEN levComp = 1
ENDIF
ENDIF
```



06 Calculate bounce angle

This section checks to see if we have hit the trampoline and adjusts our bounce accordingly. If Spikey goes below the bottom of the screen, that means he has missed the trampoline and so the lives are reduced. If the screen sides are hit, we reverse the bounce, adjust the sprite angle, then plot the sprite.

```
IF hhYdir = 0 THEN
hhY = hhY - hhGrv
IF hCol = tramp THEN
hhYdir = 1
hXdiff = (hhX - MOUSEX) / 3
IF ABS (hXdiff) > trampW / 6 THEN
hYpow = .51
hhYspd = .1
```

```
hXdiff = 0
hXpow = .1
ENDIF
hYpow = hYpow + .05
hXpow = hXpow + hXdiff / 20
hhAngle = hhAngle + hXdiff / 50
setSpriteAngle (hhID, hhAngle)
ENDIF
ENDIF
IF hhYdir = 1 THEN
hhY = hhY + hhGrv
IF hhGrv <= .5 OR hhY >= gHeight THEN hhYdir =
    0
ENDIF
IF hhY <= 0 THEN
hhLives = hhLives - 1
IF hhLives > 0 THEN
PROC newLifeVariables
PROC getready
ELSE
gameOver = 1
ENDIF
ENDIF
IF hYpow >= 1.6 THEN hYpow = 1.6
hhAngle = hhAngle + hXdiff / 50
setSpriteAngle (hhID, hhAngle)
IF hhX <= hhW / 2 OR hhX >= gWidth - hhW / 2
    THEN hXpow =  - hXpow
hhX = hhX + hXpow
plotSprite (hhID, hhX, hhY, 0)
ENDPROC
```

07 Wait for player to start

Now we enter the code so the player can position the trampoline with a mouse click.

```
DEF PROC getready
CYCLE
getMouse (a, b, mousebutton)
plotSprite (hhID, hhX, hhY, 0)
plotSprite (tramp, MOUSEX, trampY, 0)
UPDATE
REPEAT UNTIL mousebutton
ENDPROC
```

08 Move the balloons

The balloon procedure sets up an animation loop that increases the animation sequence every four frames. Our balloons will each be stored in an array, which we'll look at in the next step. It then checks to see if a balloon is in pop mode, and if so we rotate it, decrease its size and drop its Y position so that it falls down the screen, shrinking and spinning. If it is not due to be popped, we move each balloon in its direction and then check to see if it goes off the side of the screen, and if so prepare it for display on the opposite side of the screen.

```
DEF PROC balloon
bAnCtr = bAnCtr + 1
IF bAnCtr > 4 THEN
bAnID = bAnID + bAnDir
bAnCtr = 0
ENDIF
```

"If the screen sides are hit, we reverse the bounce, adjust the sprite angle, then plot the sprite"

09 Set up the balloons

The initial balloon data is essential to get things running smoothly. Here we set up an array to store all the information needed about each balloon. In this case, we have the sprite ID, X position, Y position, speed, score value, and then active and pop mode states. Compare this section with the code in the previous step to get an idea of how these variables are all labelled.

```
DEF PROC setupBals
bMinX =  - bW * 2
bMaxX = gWidth + bW * 2
bStep = (bMaxX - bMinX) / 20
ctr = 0
FOR n = bMinX TO bMaxX - bW STEP bStep CYCLE
b(ctr, 3) = gHeight - bH
b(ctr + 20, 3) = gHeight - bH * 2.5
b(ctr + 40, 3) = gHeight - bH * 4
b(ctr, 2) = n
b(ctr + 20, 2) = n
b(ctr + 40, 2) = n
b(ctr, 4) = .5
b(ctr + 20, 4) = -.2
b(ctr + 40, 4) = .1
b(ctr, 7) = 200
b(ctr + 20, 7) = 100
b(ctr + 40, 7) = 50
b(ctr, 6) = 10
b(ctr + 20, 6) = 10
b(ctr + 40, 6) = 10
b(ctr, 1) = 1
b(ctr + 20, 1) = 1
b(ctr + 40, 1) = 1
b(ctr, 8) = 0
```

```
b(ctr + 20, 8) = 0
b(ctr + 40, 8) = 0
plotSprite (b(ctr, 0), b(ctr, 2), b(ctr, 3), 0)
plotSprite (b(ctr + 20, 0), b(ctr + 20, 2),
    b(ctr + 20, 3), 0)
plotSprite (b(ctr + 40, 0), b(ctr + 40, 2),
    b(ctr + 40, 3), 0)
ctr = ctr + 1
UPDATE
REPEAT
ENDPROC
IF bAnID >= 6 OR bAnID <= 0 THEN bAnDir =  -
    bAnDir
FOR I = 0 TO 60 CYCLE
IF b(I, 8) = 1 THEN
b(I, 6) = b(I, 6) - .1
b(I, 3) = b(I, 3) - 2
b(I, 5) = b(I, 5) + 10
setSpriteSize (b(I, 0), b(I, 6) * 16)
setSpriteAngle (b(I, 0), b(I, 5))
IF b(I, 6) <= 0 THEN
hideSprite (b(I, 0))
b(I, 1) = 0
ENDIF
ENDIF
IF b(I, 1) THEN
plotSprite (b(I, 0), b(I, 2), b(I, 3), bAnID)
b(I, 2) = b(I, 2) + b(I, 4)
IF b(I, 4) < 0 THEN
IF b(I, 2) <= bMinX THEN b(I, 2) = bMaxX
ENDIF
IF b(I, 4) > 0 THEN
IF b(I, 2) >= bMaxX THEN b(I, 2) = bMinX
```

```
ENDIF
ENDIF
REPEAT
ENDPROC
```

10 Set up the game itself

The main setup section configures the screen and update settings, loads the background image and defines the main variables, including the balloon arrays and indexes that we have already seen in some of the previous steps. It is worth revisiting this section once you have finished putting Spikey Pop together, because you can test yourself by changing the size of the array and modifying the other code accordingly.

```
DEF PROC setup
setMode (1024, 600)
mouseOff
back1 = loadImage ("back1.png")
DIM b(60, 10)
balloon = 0
bAnID = 0
bAnDir = 1
bAnCtr = 0
ENDPROC
```

11 Draw the sprites

The order that sprites are created in is the order in which they will be displayed on the screen. Therefore if you want a sprite to always be on top of every other sprite, create it last.

You must set a transparent colour using setSpriteTrans (pop, 255, 0, 255). Setting the transparent

colour means you can make sure that sprites don't obscure others when they overlap.

Understanding the setSpriteOrigin command is important. The default is bottom-left, so you need to use offsets if you want to control the sprite from its middle. Far more convenient is to set the origin using: **setSpriteOrigin (pop, getSpriteW (pop) / 2** and then **getSpriteH (pop) / 2)**, as this sets the origin at the absolute centre of the sprite. Other important sprite commands include setSpriteSize, setSpriteAngle and advanceSprite – more information can be found in the Programmer's Reference Guide.

```
DEF PROC sprites
FOR n = 0 TO 60 CYCLE
b(n, 0) = newSprite (7)
FOR nn = 1 TO 7 CYCLE
num$ = STR$ (nn)
IF n >= 0 AND n <= 19 THEN loadSprite (“spe” +
    num$ + “.png”, b(n, 0), nn - 1)
IF n >= 20 AND n <= 39 THEN loadSprite (“spc”
    + num$ + “.png”, b(n, 0), nn - 1)
IF n >= 40 AND n <= 59 THEN loadSprite (“spd”
    + num$ + “.png”, b(n, 0), nn - 1)
REPEAT
REPEAT
bH = getSpriteH (b(0, 0))
bW = getSpriteW (b(0, 0))
tramp = newSprite (1)
loadSprite (“tramps.png”, tramp, 0)
trampY = -5
trampH = 20
trampW = getSpriteW (tramp)
trampL = trampW / 2
```

```
trampR = gWidth - trampW / 2
hhID = newSprite (1)
loadSprite ("hedge.png", hhID, 0)
setSpriteSize (hhID, 110)
FOR n = 0 TO hhID CYCLE
setSpriteTrans (n, 255, 0, 255)
setSpriteOrigin (n, getSpriteW (n) / 2,
    getSpriteH (n) / 2)
REPEAT
setSpriteOrigin (tramp, getSpriteW (tramp) / 2,
    0)
ENDPROC
```

12 Clean up

This final routine clears and resets all the sprites, and is needed at the end of each level and at the start of a new life.

```
DEF PROC hideSprites
FOR n = 0 TO hhID CYCLE
setSpriteSize (n, 100)
setSpriteAngle (n, 0)
hideSprite (n)
REPEAT
ENDPROC
```

FUZE BASIC V3 for Linux and the Raspberry Pi, the Programmer's Reference Guide and a Project Workbook are all freely available to download from **https://www.fuze.co.uk/lair**.

Working with RSS feeds

Learn how to build a feed ticker with your Raspberry Pi and a display to keep track of your social media feeds

There are many projects around the Internet that use the Raspberry Pi as the engine for ticker-type displays. In this way, you can keep track of all of your Twitter or Facebook feeds. This time, we will take a look at another ticker service, specifically RSS feeds. While we could start with first principles, looking at making raw network connections and parsing the returned RSS data, that is a bit beyond the scope of such a short article. Instead, we will look at using the Python module 'feedparser', or the 'Universal Feed Parser'. This module will abstract out the lower level complications and enable us to focus on actually playing with the RSS data. True to its name, the Universal Feed Parser can work with most feed formats currently in use; this includes multiple versions of RSS, multiple versions of Atom and even CDF. Installation should be easy for most people. If you are using something like Raspbian, you can install it with:

```
sudo apt-get install python-feedparser
```

If you are using something else, you can always use pip and install it directly from Pypi.
Once you have feedparser installed, you need to start by defining the feed you want to read. While feedparser can read input from a file, or even a string object, the source we are most interested in is reading the RSS data from a URL. This way, we can get access to the most recent entries. The most basic form this takes looks like:

```
import feedparser

feed1 = feedparser.parse('http://feedparser.org/
    docs/examples/atom10.xml')
```

This gives us an object that allows us to start accessing the data provided by this RSS feed. For example, you can get the title of this feed from the element feed1['feed']['title']. The feed element of the returned object contains information about the RSS channel as a whole. This includes the title, other items like the description, the feed link and the published date for the feed. This is good for the portions of your display where you indicate where a particular entry came from. The actual entries are available as a list, stored as the 'entries' element of the 'feed' object. Since this is a list, you can access the individual entries with all of the usual list syntax, such as slices. To save on typing, you may want to create a new reference to this list of entries, with something relating to:

```
entries_list = feed1['entries']
```

Each of these entry elements has a number of values available. The individual articles have a title, a link URL

Why Python?

It's the official language of the Raspberry Pi. Read the docs at https://www.python.org/doc/

and a description along with a summary of the article. In most cases, these article entries will have embedded markup of one kind or another. There is also a published date, useful for if you wanted to filter out specific articles based on the date. Depending on how you want to have the display, you could just loop through and print the article titles. Or, you may want to print out the article summaries.

Now that we have collected the information, we need to display it. If you wanted to use an LCD display to have a scrolling ticker readout, there are several modules available. The one we will use in the sample is RPLCD. This particular module was inspired by the CharLCD module from Adafruit Industries, which it wrote to support the LCD units it builds to add to your Raspberry Pi. In order to use this, you need an LCD module plugged into the GPIO pins. You can then write a string to the LCD with something resembling:

```
from RPLCD import CharLCD

lcd = CharLCD()
lcd.write_string(u'Test String')
```

This is pretty simple to use. You can just loop through your feeds, pull the article titles and summaries, then write them out to the LCD device. If you do not want to use an LCD display, or would simply rather use a more traditional display, you can use a banner module to get the same rolling ticker type of display. The classic command line utility on Linux to do this type of display is figlet. In our examples here, we will use a port of figlet called pyfiglet. You can select one of the multiple fonts

and print out the feed data to the screen. Then to use it within your Python program, you can use something like:

```
from pyfiglet import Figlet

f = Figlet(font='slant')

print f.renderText('sample text')
```

This means you can use almost anything for a display, as long as it has an HDMI connection. There is a massive number of fonts available to change and personalise the look of your very own RSS feeds. You may need to inform pyfiglet how wide the physical display is, in some number of characters. The default is 80 characters, which is the default on most terminal applications. If you are using a smaller screen, this is something you may need to think about how to deal with.

Now that we have looked at the Universal Feed Parser, you have no reason to miss any of the important news that may come your way across the internet. And since we are playing around with text and font, you can feed this news to whatever kind of display you would like. This could be something like a full monitor screen or just a single line of notifications display. The really nice thing is that you can fit your original Raspberry Pi news feed display just about anywhere you like, or build it into a more portable setup. Hopefully, this will inspire you to add your own personalised news reader to your home or workspace.

The Code OPTIMISATION

```
import feedparser  # Reading RSS feeds

# The first thing to do is to read the RSS feed
feed1 = feedparser.parse('http://feedparser.org/docs/
examples/atom10.xml')
feed1['feed']['title']  # The title of the feed is
available
feed1['feed']['link']  # You can get the originating
link for the feed
feed1['feed']['description']  # The feed has an overall
description
feed1['feed']['published']  # The publication date is
also available

articles = feed1['entries']  # The articles are
available as a list
articles[0].title  # Each article has a title
articles[0].description  # and a description
articles[0].published  # and the publication date
articles[0].summary  # The summary is probably useful,
too

# You can display the article titles on an LCD
from RPLCD import CharLCD
lcd = CharLCD()
lcd.write_string(articles[0].title)

# Or you can display on the monitor
from pyfiglet import Figlet
f = Figlet(font='slant')
print f.renderText(articles[0].title)
```

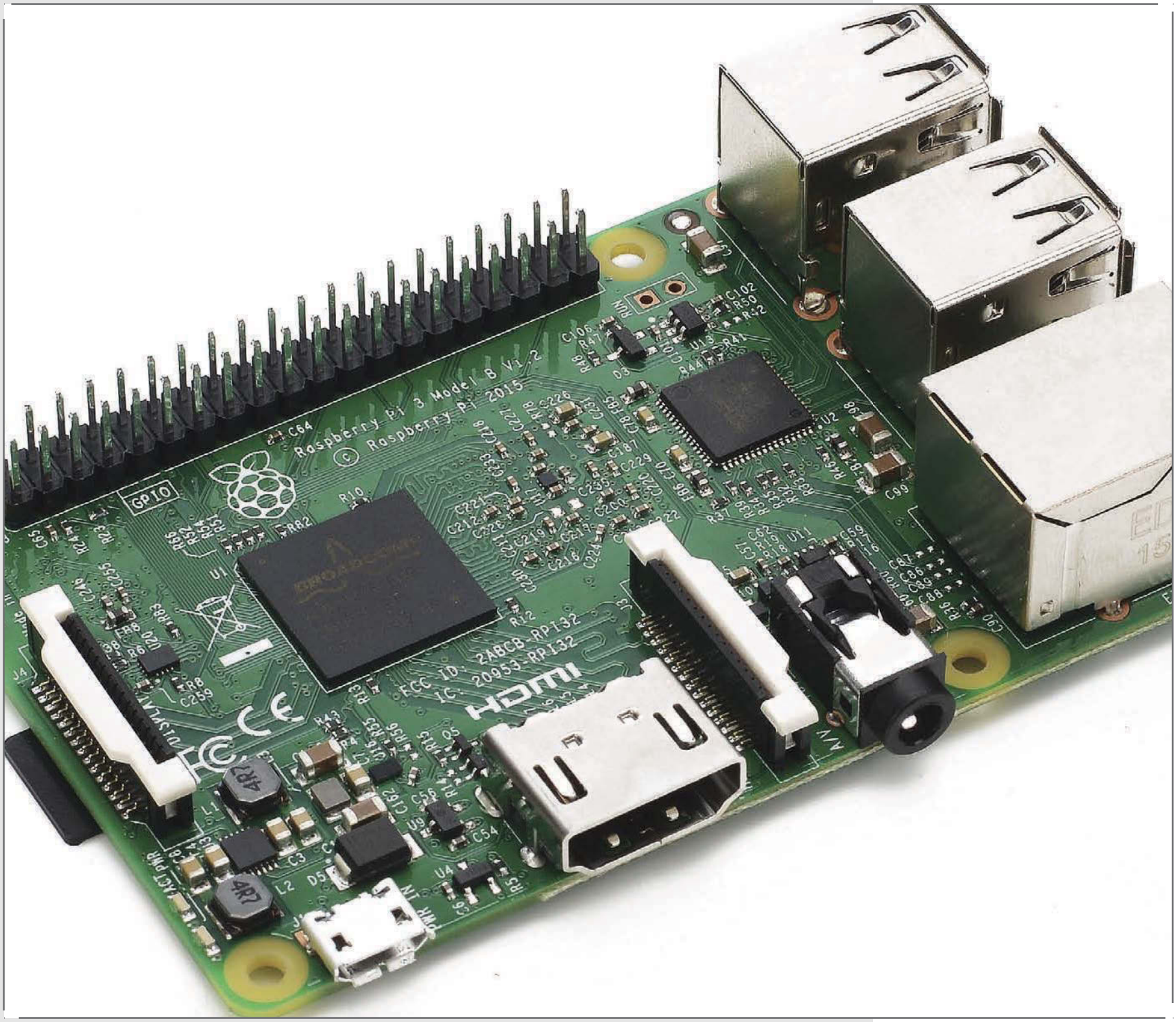
```
# Generating RSS feeds
import datetime
import PyRSS2Gen

rss = PyRSS2Gen.RSS2(
    title = “MY Raspberry Pi”,
    link = “http://www.example.com/Python/PyRSS2Gen.
html”,
    description = “This is an example RSS feed”,
    lastBuildDate = datetime.datetime.now(),
    items = [
        PyRSS2Gen.RSSItem(
            title = “Article 1”,
            link = “http://www.example.com/news/030906-
PyRSS2Gen.html”,
            description = “This is the first article”,
            guid = PyRSS2Gen.Guid(“http://www.example.com/
news/030906-PyRSS2Gen.html”),
            pubDate = datetime.datetime(2003, 9, 6, 21,
31)),
        PyRSS2Gen.RSSItem(
            title = “Article 2”,
            link = “http://www.example.com/writings/diary/
archive/2003/09/06/RSS.html”,
            description = “This is the second article”,
            guid = PyRSS2Gen.Guid(“http://www.example.com/
writings/diary/archive/2003/09/06/RSS.html”),
            pubDate = datetime.datetime(2003, 9, 6, 21,
49)),
    ])
rss.write_xml(open(“pyrss2gen.xml”, “w”))
```

Sharing your RSS feed

There's a flip side to reading RSS feeds to keep up with the news: publishing RSS from your Raspberry Pi so that everyone else can keep up with what is going on with you. While you can craft all of the required meta information that wraps your feed and all of its articles by hand, this is not necessary. You can import the Python module PyRSS2Gen, which wraps all of the formatting work for you. There is a core function, named PyRSS2Gen.RSS2(), that creates your formatted RSS feed. It takes a number of named parameters that enable you to set all of the metadata. You can set the title, a link and a description, and the parameter items takes a list of the individual article entries. Each of the items also needs to be formatted, and this is done with the function PyRSS2Gen.RSSItem(). You can set the metadata for the item with a title, link, description and a publication date. Once everything is properly formatted, you need to dump this out so that it is available to other people. The object returned by the PyRSS2Gen.RSS2() has a write_xml() function to dump the final XML file for your feed. You need to give it a file handle to write to, so you could use something like:

```
rss.write_xml(open("pyrss2gen.xml", "w"))
```

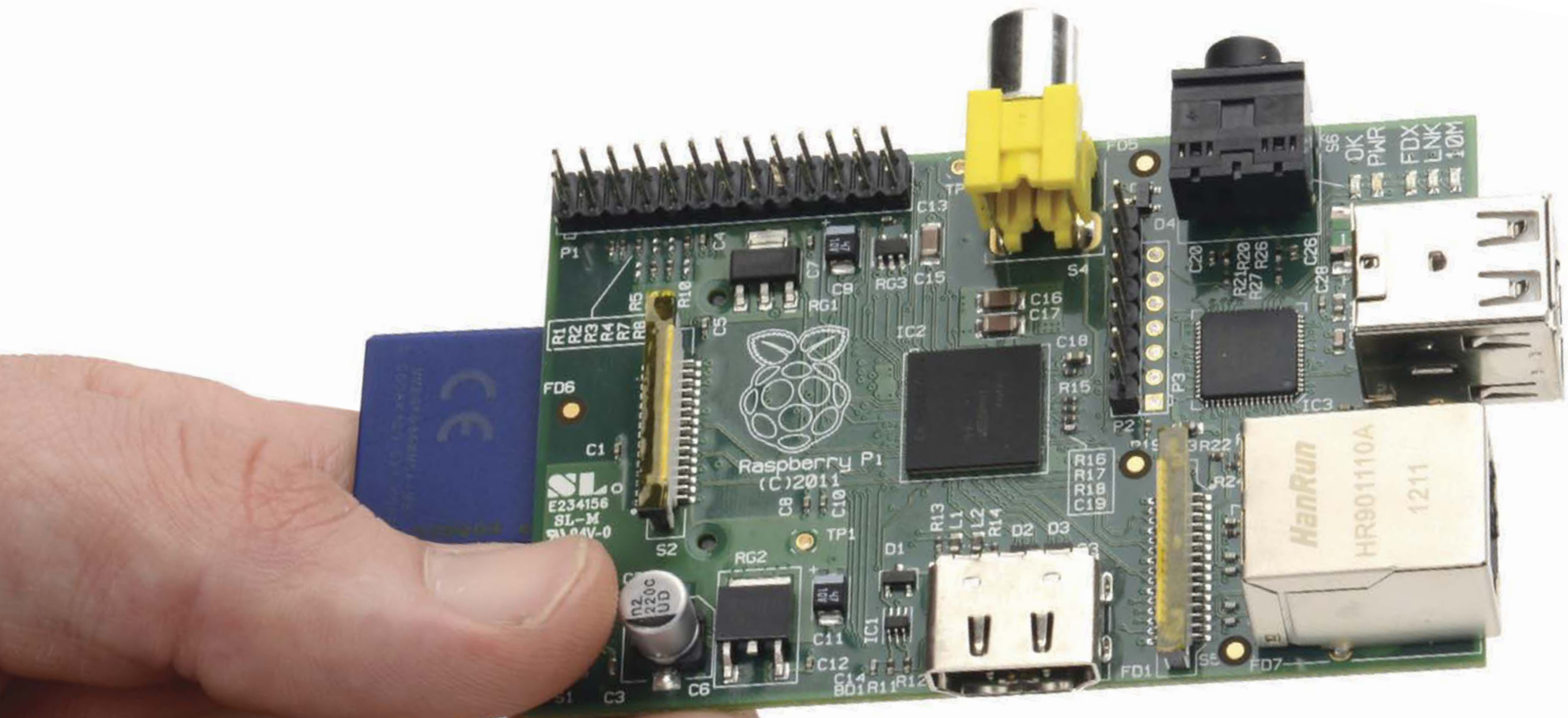
… to dump the feed contents to. If you have regular information on your Raspberry Pi that you want to provide to the outside world, you can set up a cronjob to keep it updated. You will also need to have some way to make the feed visible to the outside world. You can do this by setting up a web server or at least putting the file up in an accessible location – this might be a file service, like Dropbox or an equivalent.



Talking Pi

Join the conversation at...

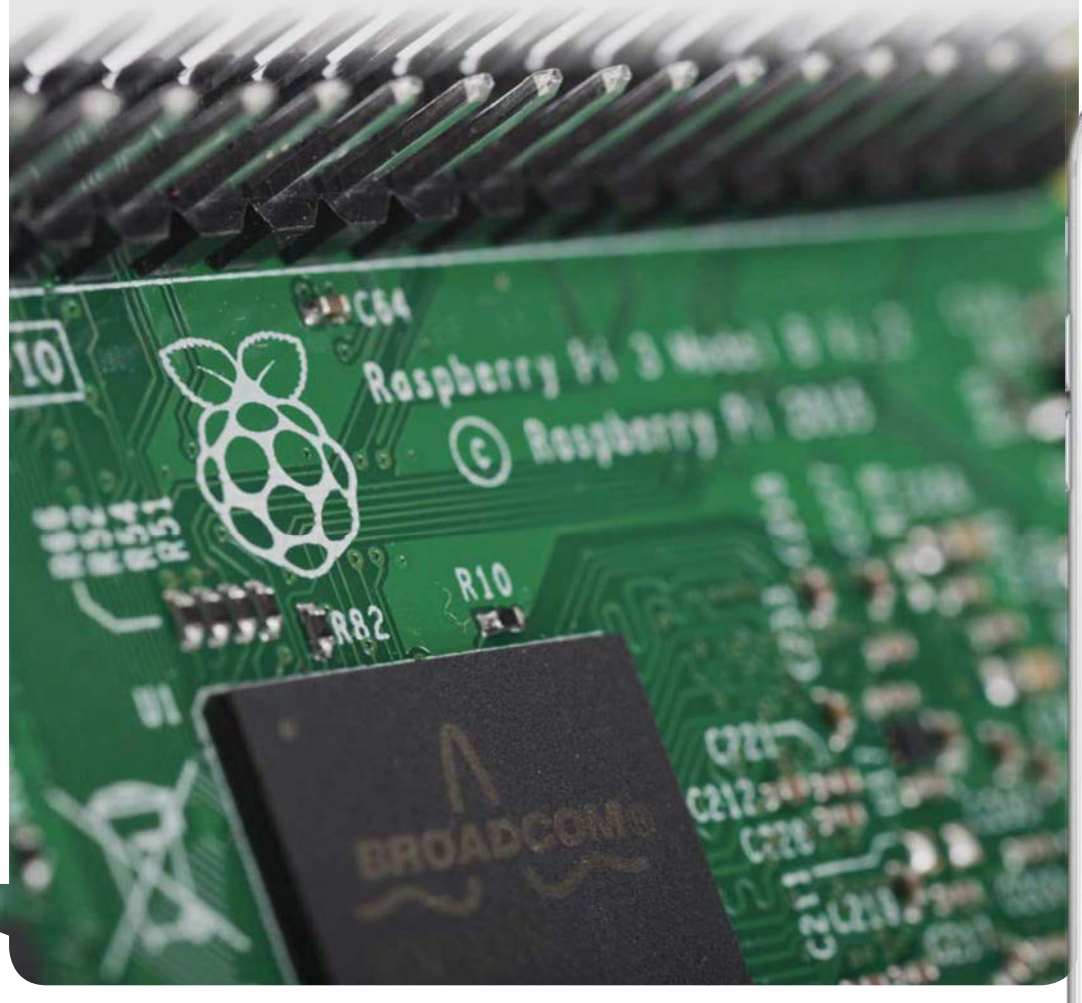
@linuxusermag | Linux User & Developer | @ RasPi@imagine-publishing.co.uk



The Raspberry Pi encourages a lot of hands-on work and this means it doesn't quite work like – or as easily as – your laser focus-tested smartphone interface, or even a normal computer. So we're answering your burning Raspberry Pi questions each issue – get in touch with us on Twitter, Facebook or by email.

One thing I never see mentioned is a cold-call blocking system? I built one with my first Pi Zero and it's great!
Brian via email

Hi Brian. Do you have five minutes to take a telephone survey about Raspberry Pi? Kidding aside though, a cold-call blocking system is a great idea. In your longer letter you mentioned the software you use (NCID; **http://ncid.sourceforge.net/**) and explained more about how your system hangs up on cold callers without the phone ringing, while calls from favourite callers are displayed on your computer or tablet so you know who's on the line. We think that this is a brilliant idea – we're plagued by robocalls about PIP, and we still get the odd chancer trying to tell us our Windows PC is compromised (LOL – Ed), so we'll definitely look into it soon!



Keep up with the latest Raspberry Pi news by following @LinuxUserMag on Twitter. Search for the hashtag #RasPiMag

JUST A SCORE
WHAT'S YOUR JUST A SCORE?

Have you heard of Just A Score? It's a new, completely free app that gives you all the latest review scores. You can score anything in the world, like and share scores, follow scorers for your favourite topics and much more. And it's really good fun!

Have you seen the cool dinosaur models that are all Pi-controlled?
Katy via email

The animatronic dinosaurs at Blackgang Chine on the UK's Isle of Wight are marvellous, aren't they? Dinosaurs had brains the size of walnuts, but these ones are a bit brighter as their brains are made out of Raspberry Pis! For those that haven't seen them, head over to http://www.blackgangchine.com to discover Blackgang Chine's Pi-powered dinos. They also host Pi Jams there – for more information, email Dr Lucy Rogers on BlackgangPi@lucyrogers.com.

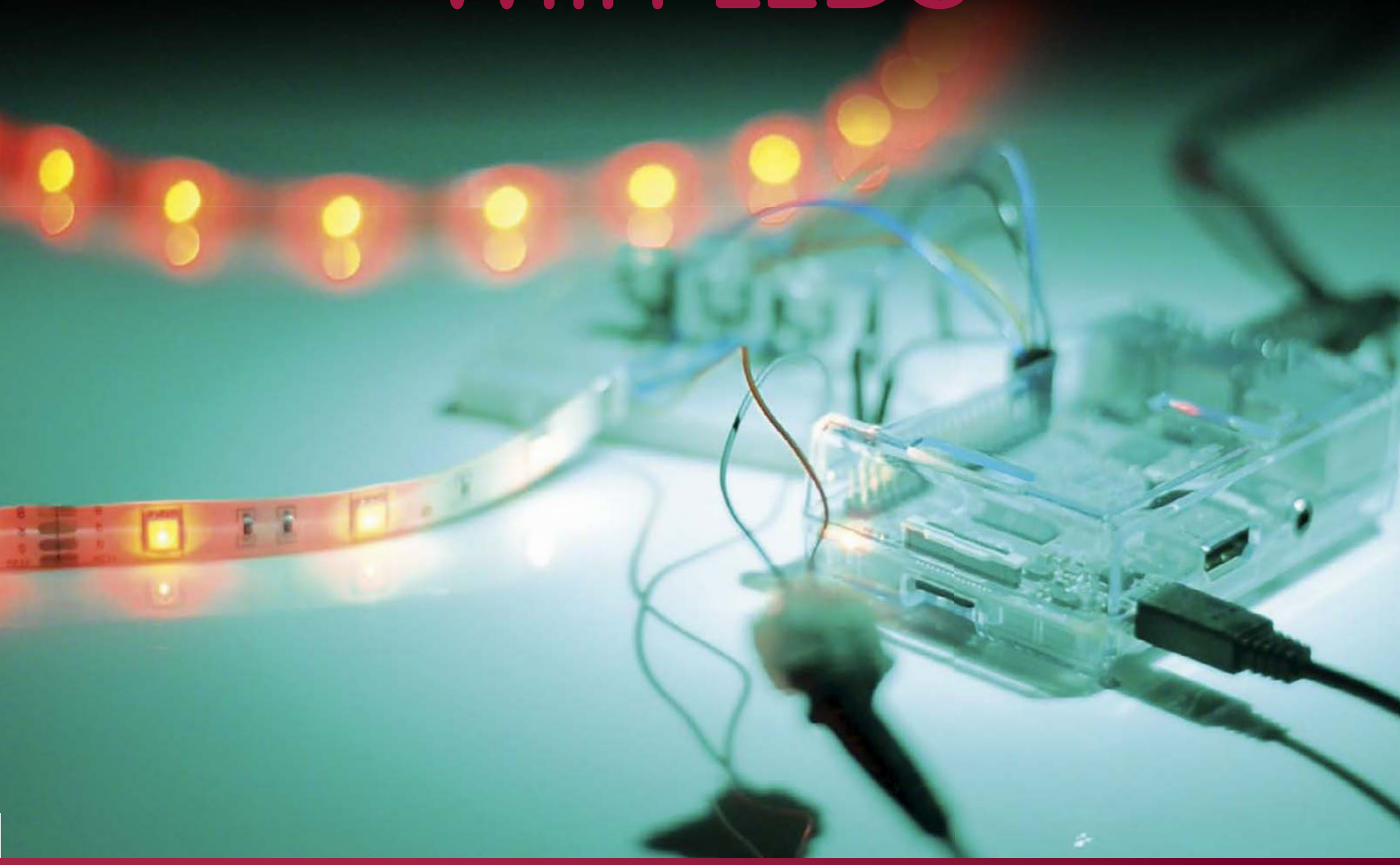
Does my Raspberry Pi need a heatsink at all?
Kev via email

The short answer is "Probably not"; the longer answer is "It depends what you're doing with it". Running alone or in most low-powered maker projects, most models of Pi should be absolutely fine without a heatsink and shouldn't get hot enough to need one, although the Raspberry Pi Foundation does recommend using a heatsink on the Model 3B if you're overclocking it (making it work faster and harder than its specifications), but then overclockers tend to put a heatsink on everything! If your project does need one, we like the Mod My Pi heatsink kit. Find it at http://bit.ly/1U8FUiQ.

Next issue

Get inspired **Expert advice** **Easy-to-follow guides**

Visualise music with LEDs



Get this issue's source code at:

www.linuxuser.co.uk/raspicode